UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Estudio, análisis y detección de ciertas anomalías en el contexto de seguridad informática

Autor: Adrián Navas Ajenjo
Tutor: Francisco de Borja Rodríguez Ortiz

MAYO 2020

Estudio, análisis y detección de ciertas anomalías en el contexto de seguridad informática

AUTOR: Adrián Navas Ajenjo
TUTOR: Francisco de Borja Rodríguez Ortiz

Escuela Politécnica Superior
Universidad Autónoma de Madrid
mayo de 2020

Resumen

A diario, y cada día más, utilizamos las Tecnologías de la Información para comunicarnos. Esto nos lleva a poner cada vez más el foco de atención en la seguridad informática, ya que, junto con la cantidad de usuarios, crece también la cantidad de ciberdelincuentes que pretenden obtener beneficio a costa de dichos usuarios. Por ello, es primordial priorizar en la detección ciberataques, para así poder defendernos de estos ciberdelincuentes. Esta detección la podemos enfocar y enmarcar dentro del campo de la detección de anomalías en los sistemas en los que hacemos uso de las Tecnologías de la Información.

En este trabajo se estudiarán algunas de las anomalías que pueden ocurrir en los sistemas, las técnicas y metodologías que existen para la detección de estas, así como la aplicación y mejora, en caso posible, de estas metodologías en diferentes bases de datos relacionadas con anomalías en el contexto de la seguridad informática.

Normalmente, el problema de detección de anomalías va ligado al problema del desbalanceo de clases ya que el número de anomalías solo representa una pequeña proporción del total de elementos. Ante este problema, se estudiarán dos enfoques diferentes para tratar las bases de datos desbalanceadas: el enfoque de clasificación de clase única y el enfoque de clasificación multiclase (binaria en este caso) aplicando técnicas de *oversampling* y *undersampling* para aumentar el número de elementos de la clase anomalía y reducir el número de elementos de la clase normal respectivamente con el fin de balancear las clases y convertir los problemas desbalanceados en problemas de clasificación binaria balanceados.

Así mismo, se propone para cada uno de los enfoques un procedimiento a seguir y los experimentos consiguientes para cada una de las bases de datos estudiadas. Estas bases de datos que se van a estudiar son la base de datos CSIC HTTP 2010 la cual está compuesta de peticiones HTTP normales y anómalas y la base de datos UNSW-NB15 la cual está compuesta de flujos de tráfico de red normales y anómalos (correspondientes con diferentes ciberataques). Al utilizar estas dos bases de datos se estudia el problema desde dos perspectivas diferentes: la capa siete y la capa cuatro del modelo OSI para así, estudiar el problema de la detección de anomalías en dos contextos completamente diferentes en el campo de la seguridad informática.

Junto a este procedimiento se estudiará la complejidad de los datos de las bases de datos estudiadas con el fin de observar la relevancia que esta tiene en la detección de anomalías.

En este trabajo se muestra que utilizar los algoritmos de *oversampling* y *undesampling* mejora considerablemente la detección de anomalías, sobre todo con los métodos de *oversampling* individualmente, y mejoran mucho más los resultados de la detección cuanto mayor es el desbalanceo entre la clase anomalía y la clase normal.

Abstract

Daily, and every day more, we use Information Technologies for communicate with others. This take us to focus on cybersecurity because with the increasing number of users, the number of cybercriminals trying to get benefit at the expense of these people increases. Because of that, it is essential to prioritize on the attack detection to defend us from these cybercriminals. We can focus this detection on the anomaly detection scope of the Information Technology system we use.

O¡In this Bachelor thesis we will study some of the anomalies that occurs on the Information Systems, detection techniques and methodologies and the application of this techniques and methodologies on different anomaly databases on the cybersecurity context.

Usually, the anomaly detection problem is related to the class imbalance problem because the number of anomalies represents just a little proportion of the total of elements. In response to this problem, we will study two different approaches to work with the databases: the One Class Classification approach and the multi-class classification approach (binary problems in this case) by using oversampling and undersampling techniques to handle the class imbalance problem and convert an imbalanced problem on a balanced binary problem. At the same time, it is proposed for each approach a procedure to follow and the corresponding tests for each studied database. These studied databases will be CSIC HTTP 2010 database formed by normal and anomalous (corresponding to web attacks) HTTP requests and the UNSW-NB15 database formed by normal and anomalous (corresponding to network attacks) network flows. With these databases the problem will be studied by two different angles: tier seven and tier four of the OSI network model and this way we can study the anomaly detection problem on two completely different contexts of the cybersecurity scope.
With this procedure, we will study the database data complexity to observe the relevancy that it has on the anomaly detection problem.
This Bachelor thesis shows that using oversampling and undersampling algorithms improve the anomaly detection and overall using oversampling methods. These methods improve more the anomalies detection when there is more class imbalance on the databases between normal and anomalous class.

Palabras clave

Detección de Anomalías, Aprendizaje Automático, Desbalanceo de clases, Ciberseguridad, Ciberataques, Complejidad de bases de datos, Técnicas de sobremuestreo y submuestreo.

Keywords

Anomaly detection, Machine Learning, Imbalanced classes, Cybersecurity, Cyberattacks, Database data complexity, Oversampling and undersampling techniques.

Agradecimientos

Primero que nada, quiero agradecer a mi familia todo el apoyo que me han ofrecido durante estos años de carrera, y con mención especial para mi madre, por ser quien más me ha apoyado durante todo este tiempo.

Finalmente, quiero agradecer a mi tutor oficial, Francisco de Borja Rodríguez y a Luis Lago, que, aunque no figure como tutor oficial de este trabajo, realmente ha sido como un segundo tutor para mí, por todo el tiempo invertido en tutorías, debates, charlas y correcciones sobre el trabajo y la temática del mismo.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Cada día, las Tecnologías de la Información están más presentes en nuestra sociedad y en la forma que tenemos de comunicarnos. Este crecimiento de uso de estas tecnologías va de la mano con el crecimiento del número de ciberdelincuentes [1] que pretenden hacer negocio a costa de los usuarios. Por esto, es primordial poner el foco de atención en la detección de estos ataques que, en el campo de la Seguridad Informática y las Comunicaciones, pueden ser tomados algunas veces como anomalías dentro de todo un inmenso volumen de tráfico normal. Por poner un ejemplo, entre los ataques más habituales están los ataques de denegación de servicio (DoS), inyección SQL o un escaneo de puertos [2][3].

Hoy en día, los mecanismos más utilizados para la detección de ataques son dos:

- Los sistemas de reglas como pueden ser los firewalls, que siempre dependen de un conocimiento previo de la arquitectura de un Sistema de Comunicaciones mediante el cual se modela el comportamiento esperado y por tanto bloquean las acciones que no entren dentro del conjunto de reglas.
- Los sistemas de firmas como pueden ser los antivirus o los Sistemas de Detección de Intrusos, que dependen del conocimiento que aportan los equipos de desarrolladores que soportan el antivirus los cuales analizan la red en busca de ejecutables maliciosos para crear las firmas y suministrarlas a los usuarios de su antivirus.

Estos sistemas, si bien muy extendidos, también son muy conocidos por los atacantes, que cada vez sofistican más sus ataques con el fin de evadir estos sistemas de protección.

Por esto, surge la necesidad de investigar otros posibles métodos a implementar para la detección de anomalías en Sistemas de Comunicaciones, y de la mano con una serie de bases de datos relacionadas con anomalías en Sistemas de Comunicaciones, se va a estudiar una serie de bases de datos con el fin de detectar el mayor número de anomalías posible en estos sistemas mediante el uso de técnicas de Aprendizaje Automático [4][5][6]. Algunas empresas también están desarrollando productos en esta línea como puede ser [7].

Es bien conocido que las técnicas de Aprendizaje Automático en clases desbalanceadas tienen ciertas peculiaridades, ya que cuando se habla de anomalías, se habla de un pequeño porcentaje del total de elementos lo cual supone una nueva necesidad de tratar los conjuntos de datos de acuerdo con este desbalanceo.

Se pretende hacer un estudio de este tipo de sistemas dónde los ataques son un problema de clasificación binario donde las clases están desbalanceadas y por esto se necesitan analizar los problemas de forma diferente a otros de Aprendizaje Automático donde las clases están más balanceadas. Esta forma incluye los dos enfoques que son tratados en este trabajo: enfoque de clasificación binaria con técnicas para resolver el desbalanceo de clases y el enfoque de clasificación de clase única o detección de anomalías.

Para realizar esta detección de anomalías, se hace uso de bases de datos relacionadas con ataques en el contexto de seguridad informática [8] las cuales disponen de diferentes tipos de ataque. En los Sistemas de Comunicaciones estos ataques pueden observarse desde dos niveles diferentes: a nivel de red (capa 4 del modelo OSI) y a nivel de aplicación (capa 7 del modelo OSI). Por eso, en este trabajo se va a estudiar este problema en ambos enfoques con el fin de observar los resultados en cada uno de ellos.

1.2 Objetivos

El objetivo fundamental es estudiar la detección de anomalías relacionadas con seguridad las cuales presentan el problema de desbalanceo de clases dónde la clase minoritaria es la clase anomalía, para ello se van a estudiar diferentes subobjetivos:

- Estudio de diferentes bases de datos de anomalías relacionadas con seguridad informática con el fin de poder contrastar los resultados obtenidos en bases de datos existentes.
- Estudiar el problema de las anomalías mediante técnicas de Aprendizaje Automático desde un enfoque de clase binaria y desde un enfoque de detección de anomalías (clase única).
- Estudiar diferentes métricas para evaluar la detección de anomalías teniendo en cuenta que las métricas habituales de Aprendizaje Automático no se deben utilizar ya que van en favor de la clase mayoritaria.
- Posteriormente, aplicar los algoritmos más apropiados para estos problemas con el fin de estudiarlos frente a las bases de datos y compararlos entre ellos.
- Estudiar la complejidad de las bases de datos que contienen las anomalías para definir cuál es el alcance de la detección de anomalías en cada una de las bases de datos poniendo en evidencia que, a mayor complejidad de datos, mayor es el problema de detección de anomalías.
- Comparar bases de datos de diferentes complejidades para observar si los métodos de corrección del desbalanceo de clases (tanto *oversampling* como *undersampling*) mantienen su eficacia y mejoran el rendimiento de la detección de anomalías independientemente de la complejidad de los datos.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Estado del arte**: dónde se comenta el estado actual de la detección de anomalías y todas las definiciones teóricas pertinentes necesarias para el trabajo incluyendo la definición de los enfoques utilizados, las bases de datos, complejidad de los datos y diferentes tipos de ataque en Sistemas de Comunicaciones.
- **Diseño**: dónde se explica el diseño implementado para el desarrollo de este trabajo, así como la definición de los algoritmos utilizados en cada uno de los enfoques y la metodología a aplicar en cada uno de ellos junto con la definición de sus fases.
- **Desarrollo, pruebas y resultados**: dónde se comentan los resultados del desarrollo de cada una de las fases junto con los resultados finales.
- **Conclusiones, discusión y trabajo futuro.**
- **Anexos**: dónde se añaden todos los datos complementarios del trabajo.

2 Estado del arte

2.1 Detección de anomalías

La detección de anomalías es un campo de estudio muy frecuente en el campo del aprendizaje automático, y como ya se ha comentado, es un gran problema que resolver en el campo de la Seguridad Informática.
La detección de anomalías, sin embargo, presenta una gran dificultad a la hora de aplicar modelos de aprendizaje automático ya que, tanto en la realidad, como en bases de datos creadas artificialmente pero que se puedan considerar fieles a la realidad, el porcentaje de elementos considerados anomalías representan un porcentaje muy pequeño con respecto al conjunto global de datos (problema de desbalanceo de clases).
A lo largo del trabajo, y en términos de clase, tomaremos los datos cuya clase es "anomalía" como datos de clase positiva (o 1) y los datos cuya clase es "normal" como datos de clase negativa (o 0). Se toma como clase positiva a la clase anomalía ya que la positiva habitualmente es la clase de interés.

2.2 Aprendizaje automático

El Aprendizaje Automático es un campo de la Ciencia de Datos el cual tiene el fin de hacer que las máquinas aprendan. Este aprendizaje puede ser de varios tipos, y en este trabajo se va a utilizar el tipo de clasificación de patrones dónde, utilizando conjuntos de datos y una serie de algoritmos, se crean modelos con el fin de predecir la clase (atributo que es el objeto de estudio de los datos) de nuevos datos.
Para este proyecto, se van a utilizar dos enfoques de clasificación diferentes tal y como proponen en [9], la clasificación multiclase y la clasificación de clase única (*One Class Classification*). Se van a emplear estos dos enfoques ya que, como se ha comentado previamente, al tener un desbalanceo de clases muy grande, podemos optar por los siguientes enfoques:

- Enfoque de clasificación multiclase aplicando técnicas de *oversampling* de la clase minoritaria (anomalía) y *undersampling* de la clase mayoritaria (normal).
- Enfoque de clasificación de clase única, utilizando la clase mayoritaria para entrenar, y detectar entradas que no se asocian al modelo creado como anomalías.

En ambos enfoques estamos solucionando el problema del desbalanceo de clases, por lo que se van a estudiar a fondo ambos y con los métodos que sean posibles dentro de cada enfoque.

2.2.1 Clasificación multiclase

Es un tipo de clasificación en el cual en la fase de entrenamiento se utilizan entradas de datos de dos o más clases, por lo que el modelo aprende a clasificar más de un tipo de clase. Este tipo de clasificación entra dentro de la clasificación supervisada ya que los algoritmos necesitan la clase a la que pertenece cada uno de los elementos que se entrenan para poder general el modelo de clasificación.
Este enfoque se define más detalladamente en la sección 2.3.1.

2.2.2 Clasificación de clase única (One Class Classification)

Es un tipo de clasificación en el cual la fase de entrenamiento solo recibe entradas de datos de un tipo de clase, en nuestro caso, la mayoritaria. De esta forma es capaz de desarrollar un modelo que englobe todos los datos recibidos en la fase de clasificación, y después, a la hora de predecir utiliza una función de decisión (diferente en cada algoritmo) mediante la cual establece una métrica de diferencia entre los nuevos datos recibidos y el modelo aprendido,

y mediante el uso de un umbral establece a partir de qué diferencia lo considera anomalía o un dato de la misma clase del modelo.
En realidad, estos algoritmos de clasificación se podrían entender como algoritmos de detección de anomalías (*outliers*), ya que su función final es la de detectar entradas de datos que no se asemejan al modelo de entrenamiento.
Este tipo de clasificación entra en el campo de la clasificación no supervisada ya que solo se entrena utilizando una clase, solo distingue entre elementos normales y anomalías.
Este enfoque se define más detalladamente en la sección 2.3.2.

2.2.3 Medidas para la evaluación de clasificadores

Para estudiar el rendimiento de cualquier clasificador al validarlo con los datos de prueba, se utilizan una serie de métricas. Al estar ante problemas de clasificación binaria clase anomalía (o +) o clase normal (o -), tras validar el clasificador obtendremos los cuatro valores a continuación pertenecientes a la matriz de confusión:

- Verdaderos positivos (TP): número de elementos cuya clase real es positiva (anomalías) y que el clasificador ha clasificado como positivos.
- Falsos positivos (FP): número de elementos cuya clase real es negativa (normal) y que el clasificador ha clasificado como positivos.
- Verdaderos negativos (TN): número de elementos cuya clase real es negativa (normal) y que el clasificador ha clasificado como negativos.
- Falsos negativos (FN): número de elementos cuya clase real es positiva (anomalía) y que el clasificador ha clasificado como negativos.

La Tabla 1 a continuación muestra gráficamente el concepto de la matriz de confusión.

Clase predicha Clase real	Clase Anomalía (+)	Clase Normal (-)
Clase Anomalía (+)	TP	FN
Clase Normal (-)	FP	TN

Tabla 1. Matriz de confusión.

Tras haber definido los conceptos previos (correspondientes a los valores de una matriz de confusión en un problema de clasificación binario), se definen las métricas más comunes para medir el rendimiento de los clasificadores:

- *Accuracy:* Porcentaje de aciertos global.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}. \quad \textbf{(1)}$$

- Precisión: Número de positivos reales entre el número total de positivos clasificados.

$$Precisión = \frac{TP}{TP + FP}. \quad \textbf{(2)}$$

- *Recall*, sensibilidad o proporción de verdaderos positivos: Porcentaje de positivos reales detectados como tal.

$$Recall = \frac{TP}{TP + FN}. \quad \textbf{(3)}$$

- *F1Score:* Media armónica entre la precisión y la sensibilidad.

$$F1Score = 2 * \left(\frac{Precision * Recall}{Precision + Recall}\right). \quad (4)$$

- Proporción de falsos positivos: Número de casos negativos clasificados como positivos entre el total de negativos reales.

$$FPR = \frac{FP}{TN + FP}. \quad (5)$$

- Especificidad o proporción de verdaderos negativos: Número de casos negativos clasificados como tal frente al de negativos reales.

$$TNR = \frac{TN}{TN + FP}. \quad (6)$$

- ROC AUC: La curva ROC (Característica Operativa del Receptor) se trata de una curva utilizada para medir el rendimiento de un modelo de clasificación mediante la representación gráfica del TPR frente al FPR para diferentes umbrales de clasificación.
  La medida ROC AUC se trata del área debajo de la curva ROC y su valor está en el rango [0, 1] dónde 1 es el mejor valor posible.

En la sección 3.1 se comenta detalladamente cómo se aplican estas medidas en el contexto de detección de anomalías y en clases desbalanceadas.

2.2.4 Oversampling y undersampling (sobremuestreo y submuestreo)

En conjuntos de datos muy desbalanceados, los resultados de aplicar clasificadores multiclase pueden ser muy malos a la hora de modelar y detectar los datos de la clase minoritaria ya que no existen datos suficientes para que el modelo sea capaz de aprender a separarlo, mientras que la otra clase se encuentra muy favorecida. Por esto, se han desarrollado una serie de algoritmos que pueden ayudar a paliar el efecto del desbalanceo bien aumentando el número de elementos de la clase minoritaria o bien eliminando elementos de la clase mayoritaria, siempre siguiendo un algoritmo para elegir qué elementos añadir o eliminar, con el fin de equipararlas en número (o disponer de una proporción mejor entre la clase minoritaria y la mayoritaria).
En el apartado 2.3.1 se definen detalladamente los diferentes métodos utilizados.

2.3 Distintos enfoques de detección de anomalías

Se van a aplicar dos enfoques diferentes de detección de anomalías con el fin de comprobar el rendimiento de cada uno de ellos, aplicando medidas mitigantes del desbalanceo de clases existente en los datos de anomalías.

2.3.1 Clasificación multiclase

Al estar ante un tipo de clasificación que requiere de más de una clase a la hora de entrenar, el desbalanceo de clases aporta una gran dificultad a la hora de clasificar y detectar correctamente las anomalías. Por eso, para este enfoque se van a utilizar medidas de *undersampling* y *oversampling* con el fin de equiparar en número la proporción de cada clase en los conjuntos de datos. Para ello se van a considerar los siguientes algoritmos:

2.3.1.1 Algoritmos de Oversampling

Estos algoritmos aumentan el número de elementos de una clase en un conjunto de datos utilizando diferentes métodos como pueden ser la generación aleatoria o la generación sintética. Los algoritmos utilizados en este trabajo son los siguientes:

2.3.1.1.1 SMOTE

O *Synthetic Minority Over-sampling TEchnique*, se trata de una técnica de *oversampling* [27][28] utilizada para crear elementos de una clase (generalmente la clase minoritaria) de forma sintética.

Para la generación de elementos de forma sintética, se hace uso del parámetro de la proporción del número de elementos de la clase minoritaria y la clase mayoritaria, y con esta proporción se obtiene el número de elementos (N) de la clase minoritaria que se necesitan generar para obtener dicha proporción.

El algoritmo para cada uno de los puntos de la clase que se quiere ampliar selecciona los N-vecinos más cercanos (con N igual al número entero obtenido previamente), y para cada uno de ellos selecciona un punto aleatorio en el vector entre el elemento en cuestión y el vecino para generar el nuevo punto sintético. Esta elección aleatoria se realiza obteniendo un número aleatorio entre 0 y 1 y multiplicándolo por el vector diferencia de ambos puntos obteniendo así el punto exacto dónde se generará el nuevo elemento.

Un ejemplo del funcionamiento del algoritmo es el correspondiente a la Figura 1.

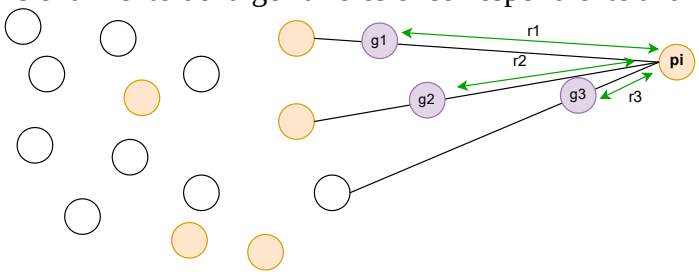

Figura 1. Ejemplo de aplicación de SMOTE.

En el ejemplo, podemos ver cómo en un conjunto de datos dónde los puntos blancos son elementos normales y los naranjas son elementos anómalos, se generan 3 elementos sintéticos (g_1, g_2 y g_3, puntos morados en la Figura 1) para el punto p_i. Estos tres puntos han sido generados seleccionando los 3 vecinos más próximos al punto, y eligiendo un número aleatorio entre 0 y 1 y multiplicándolo por el vector entre el punto p_i y cada uno de sus 3 vecinos más cercanos obteniendo las distancias r_1, r_2 y r_3 que corresponden a los espacios dónde se generan cada uno de los puntos g_i.

2.3.1.1.2 ADASYN

O *Adaptative Synthetic*, se trata de una técnica de *oversampling* [28][30] que sigue el mismo planteamiento que SMOTE, pero con pequeñas diferencias para intentar que los ejemplos generados sintéticamente sean más realistas.

Para esto, una vez obtenidos los N vecinos de cada punto de la clase minoritaria, se calcula para cada punto i:

$$r_i = \frac{\Delta_i}{N},$$

dónde Δ_i es el número de vecinos dentro de sus N más cercanos del punto i que pertenecen a la clase mayoritaria.

Tras esto, se normaliza r_i para obtener:

$$\hat{r}_i = \frac{r_i}{\sum_i r_i}.$$

Y de esta forma tener $\hat{r}_i$ como una distribución de probabilidad con la probabilidad de generar más o menos ejemplos en cada uno de los puntos de la clase minoritaria.

Una vez disponemos de $\hat{r}_i$, para cada uno de los puntos i obtenemos:

$$g_i = \hat{r}_i * N ,$$

que indicará el número de ejemplos sintéticos a crear en cada punto de la clase minoritaria. Con este valor, se escogen g_i vecinos próximos del punto i de forma aleatoria, y siguiendo el mismo mecanismo de SMOTE de seleccionar un número aleatorio entre 0 y 1 y se multiplica por la diferencia entre el punto original y el vecino escogido para así añadir el nuevo elemento sintético.
De esta forma, se pretende favorecer la creación de elementos en zonas con menor diferencia entre clase minoritaria y clase mayoritaria y así favorecer a la creación de una frontera más clara.

2.3.1.1.3 Borderline-SMOTE1

Método de *oversampling* [28][31] basado en la idea de generación de elementos de forma sintética de SMOTE, pero centrado, al igual que ADASYN, en la idea de favorecer la creación de elementos de tal forma que se obtenga una frontera de decisión más clara.
Para cada punto de la clase minoritaria, y haciendo uso de la variable N al igual que en los algoritmos anteriores, se obtienen los N vecinos más próximos. De estos vecinos, el número m' indica el número de vecinos dentro de esos N que son de la clase mayoritaria.
Una vez obtenidos los N vecinos, cada uno de los puntos p_i de la clase minoritaria se clasifica en una de estas tres clases:

- Ruido si $m' = N$
- Seguro o *safe* si $0 \leq m' < \frac{N}{2}$
- Peligro o *danger* si $\frac{N}{2} \leq m' < N$

Si el punto es considerado como ruido o como seguro, no se utiliza en el algoritmo ya que utilizarlos no ayudaría en la creación de una frontera más determinada. Por esto, los únicos puntos que se tendrán en cuenta son los puntos de peligro.
Para generar los nuevos elementos, para cada uno de los puntos de peligro (conjunto D), seleccionamos un número aleatorio de sus N-vecinos próximos, y al igual que en los dos algoritmos anteriores, se genera un nuevo elemento entre el punto y su vecino multiplicando el vector diferencia por un número aleatorio entre 0 y 1.

2.3.1.1.4 Random Oversampler

Con este método de *oversampling*, se generan de forma aleatoria copias de los elementos de la clase minoritaria hasta alcanzar el número de elementos deseados. Este método no favorece a la creación de una frontera de decisión más clara, simplemente le da más peso a la clase minoritaria ya que genera copias exactas de elementos ya existentes.

2.3.1.1 Algoritmos de Undersampling

Estos algoritmos disminuyen el número de elementos de una clase en un conjunto de datos utilizando diferentes métodos como puede ser la eliminación aleatoria. Los algoritmos utilizados en este trabajo son los siguientes:

2.3.1.1.1 Random Undersampling

Se trata de un método de *undersampling* que elimina elementos de la clase mayoritaria aleatoriamente hasta alcanzar el balance deseado entre clases.

2.3.1.1.2 Near Miss

Se trata de un método de *undersampling* [32] basada en vecinos próximos. Mediante el uso de N vecinos próximos, se seleccionan los K puntos con clase mayoritaria cuya distancia media a sus N vecinos próximos con clase mayoritaria sea la menor y se eliminan. En este

método, K es el número total de elementos que se tienen que eliminar para conseguir el balanceo entre clases requerido.

2.3.1.1 Algoritmos de Oversampling y Undersampling conjuntos

Estos algoritmos combinan métodos de *oversampling* para aumentar el número de elementos de una clase y métodos de *undersampling* para eliminar otros elementos. Los algoritmos utilizados en este trabajo son los siguientes:

2.3.1.1.1 SMOTE + ENN

Se trata de un método de *oversampling* y *undersampling* [28][33] que aplica SMOTE para generar elementos de forma sintética de la clase minoritaria, y tras esto, aplica un proceso de limpieza de los datos mediante el uso de la regla *Wilson's Edited Nearest Neighbour (ENN)* [29] y elimina cualquier elemento (tanto de clase mayoritaria como clase minoritaria) que tenga dos de sus tres vecinos próximos cuya clase sea diferente a la del punto.

2.3.1.1.2 SMOTE + Tomek Links

Se trata de un método de *oversampling* y *undersampling* [28][34] que al igual que el anterior, primero aplica SMOTE para generar elementos de la clase minoritaria, y después busca todos los *Tomek Links* y los elimina. Un *Tomek Link* es un par de elementos en el conjunto de datos cuyas clases son diferentes (uno clase mayoritaria y otro minoritaria) y cuyo punto más cercano de ambos elementos es el otro.

2.3.1.2 Algoritmos de clasificación multiclase

Y una vez aplicados estos algoritmos, se utilizarán los siguientes clasificadores multiclase ya que, tras aplicar un estudio con diferentes clasificadores en las bases de datos utilizadas, son los que mejores resultados han aportado.

2.3.1.2.1 Árboles de decisión

Se trata de un algoritmo de clasificación [35] con el cual se genera un árbol de binario utilizado para predecir clases de elementos nuevos. En el entrenamiento se genera este árbol de decisión en el cual cada uno de los nodos es una "pregunta" y cada rama de ese nodo es una respuesta que separa el conjunto de datos en otros dos subconjuntos y a su vez los nodos hijo de cada nodo son otras nuevas preguntas hasta encontrar una separación en la cual solo existen elementos de una clase y ese nodo se establece como una hoja en la cual se establece una decisión final. Este proceso se realiza de forma iterada hasta tener todo el conjunto de datos dividido en hojas finales. En el proceso de generación de cada nodo, se contempla el conjunto de datos que ha llegado hasta él y se decide cual será la mejor división en ese nodo en base a un criterio, los posibles criterios son:

- Impureza de Gini. Con este criterio se calcula la probabilidad de clasificar un elemento incorrectamente si se clasifica de forma aleatoria utilizando la distribución de clases en el conjunto de datos en el nodo en cuestión. Este valor se calcula de la siguiente manera:

$$G(C) = 1 - \sum_{c \in C} p_c^2 \,, \tag{7}$$

  dónde:

  - $p_c$es la probabilidad de la clase c en el conjunto generado por C.
  - C es la lista de clases resultante de realizar una separación.

  Una vez calculada la entropía de cada uno de los dos subconjuntos que deja el espacio, G_1y G_2, se calcula la métrica Gini de la siguiente manera:

$$G_{split} = G_1 * N_1 + G_2 * N_2 ,$$

dónde N_1 y N_2 son el número de elementos totales en cada una de las dos separaciones.

Tras esto, se selecciona la separación cuyo Gini sea menor.

- Ganancia de información. Utiliza el mismo concepto que en el paso anterior, pero con la métrica de la entropía:

$$E(C) = -\sum_{c \in C} p_c * log_2 p_c , \quad \textbf{(8)}$$

y tras calcular la entropía en cada una de las separaciones posibles, se calcula la entropía general de esa separación haciendo uso de E_1y E_2:

$$E_{split} = E_1 * \frac{N_1}{N} + E_2 * \frac{N_2}{N} .$$

Tras esto, se selecciona la separación con mayor entropía como mejor separación posible.

En este trabajo, este algoritmo se ha utilizado con dos fines diferentes: cómo clasificador para detección de anomalías en el enfoque de clasificación binaria, y cómo método de selección de atributos en una base de datos (2.4.2) utilizando la propiedad de la importancia de cada uno de los atributos calculada cómo la importancia normalizada de Gini utilizando dicho criterio para cada uno de los atributos.

2.3.1.2.2 Random Forest

Se trata de un conjunto de clasificadores por voto [36] mediante la creación de un conjunto de árboles de decisión aleatorios que aplica el concepto de *bagging* y se basa en la suposición de que la generación de árboles no correlacionados, si bien aporta mucho ruido a la clasificación al utilizar árboles aleatorios, hace que los árboles generados sean imparciales.

El algoritmo genera un número N de árboles de decisión y para generar cada uno de ellos, se entrenan con un número aleatorio de elementos de los datos de entrenamiento y con un número aleatorio de atributos de esos datos, para de esta manera, generar un bosque lo más variado posible.

Finalmente, cuando se quiere clasificar un elemento, se clasifica con todos los árboles de decisión que componen el bosque y se clasifica mediante voto como la clase más elegida por los árboles de decisión.

2.3.1.2.3 Vecinos próximos

Vecinos próximos se trata de un algoritmo de clasificación [37] basado en la suposición de que un elemento es de cierta clase si la mayoría de sus vecinos más próximos lo son.

Este algoritmo necesita un parámetro de entrada N que es el número de vecinos más próximos que se tendrán en cuenta y una métrica de distancia que será la utilizada para calcular la distancia entre dos elementos del conjunto de datos.

A la hora de clasificar, se calculan las distancias desde el elemento a calcular y todos los elementos en el conjunto de entrenamiento, se seleccionan los N más cercanos, y se observa la clase de todos ellos. Después, se calcula la probabilidad de cada clase de la siguiente manera:

$$p_c = k_c / N ,$$

dónde:

- p_c es la probabilidad de que el elemento sea de la clase c.
- k_c es el número de elementos de clase c en el conjunto K.
- K es el conjunto de los N vecinos más próximos del punto.
- N es el número de vecinos contemplados en el algoritmo.

Y finalmente, se clasifica el elemento con la clase cuya probabilidad sea mayor:
Debido a esto, para los conjuntos de datos con clase binaria (o número par de clases), es recomendable elegir un número N impar para que, a la hora de clasificar, las probabilidades de dos o más clases sean iguales y de esta forma nunca tener que realizar decisiones aleatorias.

2.3.2 Clasificación de clase única (*One Class Classification*)

Este otro enfoque, solo necesita entrenar con elementos de una única clase, y crea un modelo que se asemeja a dicha clase. Después, a la hora de clasificar, utiliza métricas de distancia (diferencia) con el modelo generado en el entrenamiento, y en base a un umbral, decide si se considera un valor que entra dentro del modelo (*inlier*) o un valor anómalo (*outlier*). Si utilizamos este enfoque, resolvemos de golpe el problema del desbalanceo de clases ya que por muy desbalanceado que sea nuestro conjunto de datos, al solo necesitar entrenar con elementos de una única clase, podemos entrenar el modelo con los elementos de la clase "normal" y después observar qué elementos se salen del modelo aprendido y así poder detectar nuestras anomalías. Para comprobar qué elementos se salen del modelo, los algoritmos utilizan una función llamada función de decisión mediante la cual establecen en función al modelo de datos de entrenamiento, un valor (también llamado *ranking*) a cada elemento que reciban el cual establece cómo de parecido o diferente es el nuevo elemento con respecto a los datos de entrenamiento. De esta forma, tras tener el modelo entrenado se puede hacer uso de cierto umbral para establecer a partir de qué valores de *ranking* se considera un elemento anómalo.
Se van a utilizar los siguientes algoritmos de *One Class Classification* para observar sus resultados:

2.3.2.1 Isolation Forest

Se trata de un algoritmo de *One Class Classification* cuyo fin es aislar las anomalías basándose en la premisa de que las anomalías son "*pocas y diferentes*" [38][39] y por esto son fáciles de aislar.
El bosque de este algoritmo está compuesto por un conjunto de árboles de decisión llamados árboles de aislamiento. Cada uno de estos árboles está generado por la división de forma recursiva del conjunto de datos de entrenamiento utilizando un atributo q aleatorio y división por ese atributo en un valor p aleatorio hasta que todos los elementos del conjunto de datos estén aislados, es decir, hasta que solo se disponga de un elemento por nodo final. Este algoritmo modela el conjunto de datos siguiendo la afirmación de que cuanto más fácil sea aislar un elemento (número de particiones necesarias para aislarlo), más probabilidad de ser una anomalía tiene.
Cómo función de decisión o de *ranking*, este algoritmo utiliza un concepto llamado *path lenght* (longitud del camino) o h(x) y que se define como el número de aristas que recorre un punto x en un *Isolation Tree* desde el nodo raíz hasta llegar a un nodo hoja final.
Para la generación del bosque de árboles, se generan los t árboles de aislamiento indicados utilizando una partición de los datos de entrenamiento de tamaño w indicado y se almacenan en la estructura del bosque.
Para la clasificación en un *Isolation Forest*, se hace uso del *score* de anomalía que tiene cada elemento basándose en el *path length* de todos sus árboles. Este *score* se calcula de la siguiente manera:

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}, \quad \textbf{(9)}$$

dónde:

- x es el punto del que queremos obtener el *score* de anomalía.
- $E\big(h(x)\big)$ es la media de *path length* de x en todos los árboles del bosque.
- $c(n)$ se define cómo la media de $h(x)$ dado un conjunto de n instancias y se calcula de la siguiente manera

$$c(n) = 2H(n-1) - \left(2 * \frac{n-1}{n}\right), \quad \textbf{(10)}$$

y donde $H(i)$ es el número armónico de i y dónde n es el número de instancias que tienen los árboles del bosque.

Finalmente, los elementos son clasificados como anomalías o elementos normales basándose en su *score* de anomalía. Cuanto más se acerque el *score* a 1 mayor será su probabilidad de ser una anomalía.

2.3.2.2 Elliptic Envelope

Se trata de un algoritmo [40] que asume que los datos normales están englobados dentro de una distribución conocida, en este caso una distribución Gaussiana. Siguiendo esta suposición, es sencillo observar qué datos que reciba el algoritmo son anomalías y qué datos no lo son ya que al conocer si distribución de probabilidad se conoce cómo de parecido es con respecto a los datos normales conocidos.

Para calcular la distancia de un elemento con respecto a la distribución Gaussiana se puede utilizar la distancia de Malahanobis que se calcula de la siguiente forma:

$$d_{\mu,\Sigma}(p) = \sqrt{(p-\mu)'\Sigma^{-1}(p-\mu)}\,, \quad \textbf{(11)}$$

dónde:

- **p** es el vector del elemento en cuestión.
- μ es el vector medio.
- Σ es la matriz de covarianza.

Y para realizar la estimación de los parámetros μ y Σ, se utiliza el estimador *Minimum Covariance Determinant* (MCD) propuesto en [41] ya que se trata de un estimador que es poco sensible a las anomalías y que permite realizar estimaciones robustas en conjuntos de datos con mucho ruido presente (elementos normales que se pueden considerar anomalías según la distribución).

2.3.2.3 Local Outlier Factor

Se trata de un algoritmo [42] basado en la densidad local de cada punto calculada de forma aproximada usando la distancia a sus N vecinos más próximos.

La idea principal que utiliza este algoritmo es que los elementos normales o *inliers* tendrán densidades locales muy altas ya que sus puntos estarán muy próximos en el espacio y las anomalías tendrán densidades locales más bajas ya que serán puntos aislados en términos generales.

Para calcular el *score* de cada punto, necesitamos tener en cuenta los siguientes conceptos:

Distance Reachability o distancia de alcance, que entre un punto A y otro punto p es la distancia máxima entre la distancia de A a su k-vecino más próximo y la distancia entre A y p.

$$reach - dist_k(A,p) = \max\{dist_k(A), dist(A,p)\}, \quad \textbf{(12)}$$

dónde:

- k es el número de vecinos más próximos que se tienen en cuenta en el algoritmo.

- $dist_k(A)$ es la distancia al punto más lejano entre los k vecinos más próximos de A.
- $dist(A, B)$ es la distancia entre los puntos A y B.

Local reachability density o densidad de alcance local que se calcula de la siguiente manera:

$$lrd_k(A) = \frac{|N_k(A)|}{\sum_{p \in N_k(A)} reach - dist_k(A, p)}, \quad \textbf{(13)}$$

dónde:

- $N_k(A)$ es el conjunto con los k vecinos más próximos de A.
- $|N_k(A)|$ es el número de puntos en ese conjunto

Y una vez tenidos en cuenta estos conceptos, se define el *local outlier factor* como:

$$LOF_k(A) = \frac{\sum_{p \in N_k(A)} lrd_k(p)}{|N_k(A)| * lrd_k(A)}. \quad \textbf{(14)}$$

Este algoritmo modela entonces la clase normal (o benigna) como los elementos con una alta densidad.

Y una vez calculado este valor para un punto p, si es mayor que 1 indica que hemos encontrado un *outlier* y si es menor de 1, que hemos encontrado un *inlier*.

2.3.2.4 Autoencoder

Un *autoencoder* [43] es una red neuronal simétrica basada en el concepto de compresión y descompresión de la información. Su entrenamiento se realiza de forma no supervisada, con datos de una clase específica que queremos modelar, y el fin es el de minimizar los errores de reconstrucción.

El diagrama general de un autoencoder es el que se define a continuación.

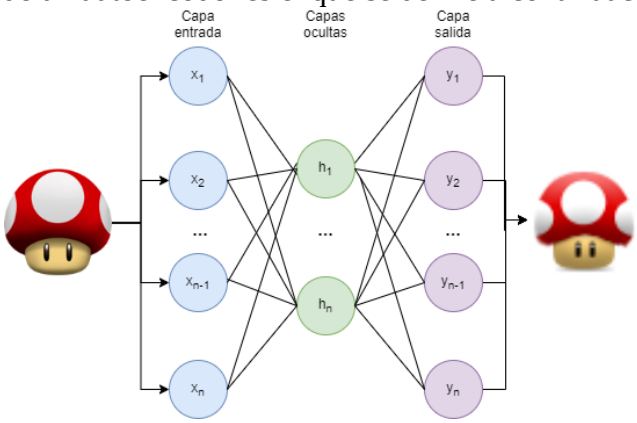

Figura 2. Diagrama general de un *autoencoder*

El concepto en el que se basan los *autoencoders* [43][44] para clasificar tras la fase de entrenamiento es el concepto de que, si hemos encontrado una forma de modelar el conjunto de datos con un error de reconstrucción bajo, cualquier elemento que no sea de la clase entrenada generará un error de reconstrucción alto y mediante el uso de umbrales en los errores de reconstrucción podemos clasificar las anomalías detectadas.

Como métrica de error de reconstrucción se utilizará el error cuadrático medio que se calcula de la siguiente manera:

$$MSE(y, \hat{y}) = \frac{1}{n} * \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \quad \textbf{(15)}$$

dónde:

- y es el vector real que entra al autoencoder.

- $\hat{y}$ es el vector de salida del autoencoder.
- n es el número de atributos que componen el vector.

Las funciones de activación utilizadas en redes neuronales para cada capa son varias, pero las utilizadas en este trabajo son las siguientes:

Tangente hiperbólica (tanh)

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \quad \textbf{(16)}$$

Unidad lineal rectificada (ReLU)

$$relu(x) = \max\{0; x\}. \quad \textbf{(17)}$$

2.4 Bases de datos

El primer objeto de estudio, son una serie de bases de datos existentes con relación a anomalías en el contexto de la seguridad informática. Estas anomalías son de diferentes tipos y ámbitos como pueden ser peticiones HTTP de clientes normales (clase "normal") y peticiones de clientes maliciosos que intentan realizar algún tipo de ataque en el servidor HTTP (clase "anomalía") o paquetes de red normales en un sistema (clase "normal") y paquetes maliciosos que intentan atacar el sistema o la integridad de este, cómo, por ejemplo, ataques de denegación de servicio (clase "anomalía").

El gran problema de este tipo de bases de datos es su escasez, ya que la generación de estas es muy complicada debido al problema de etiquetar el tráfico real. Por esto, la gran mayoría de bases de datos están creadas a partir de tráfico generado artificialmente, lo cual crea un posible problema de no asemejar al cien por cien la realidad, pero le da el valor añadido de estar tener todas las clases correctamente identificadas y etiquetadas.

No obstante, del conjunto de bases existentes disponibles se ha optado por las dos bases de datos detalladas a continuación por tratarse de dos bases de datos sin una estructura compartida dónde una de ellas (2.4.1) obtiene la información en la capa 7 del modelo OSI [1], nivel de Aplicación, a través de peticiones HTTP detalladas en el Anexo A.B y la otra (2.4.2) obtiene la información en la capa 3 del modelo OSI [1], nivel de red, a través de flujos de tráfico de red detallados en el Anexo A.C. De esta forma, es posible examinar dos problemas completamente diferentes, aunque ambas compartan el contexto de anomalías en Sistemas de Comunicaciones.

Además, ambas bases de datos disponen de complejidad de datos muy diferente (una baja y otra alta), por lo que el problema se estudia tanto para dos problemas completamente diferentes dónde las anomalías a detectar son diferentes, como para dos problemas con una complejidad diferente.

2.4.1 CSIC HTTP 2010

Se trata de una base de datos creada por Carmen Torrano [11] compuesta por una serie de peticiones HTTP con el contenido completo de cada una de ellas y etiquetadas con la clase 'normal' y 'anomalía' en función del tipo de petición.

Todas las peticiones de esta base de datos han sido generadas mediante tráfico generado artificialmente contra una aplicación web que simula una tienda creada para este propósito. Esta aplicación web se trata de una simulación de una tienda, desarrollada en JSP y ejecutado por Apache Tomcat y si bien no fue expuesta en Internet, se expuso de forma local simulando en la medida de lo posible un entorno real en Internet. Tras esto, se hizo uso de bases de datos para crear dos diccionarios, uno para crear las peticiones normales y otro para crear las peticiones anómalas y si bien las peticiones se generaron de forma artificial, las bases de datos utilizadas eran reales para adecuarlo en la medida de lo posible al máximo a la realidad.

Las entradas etiquetadas como anomalías contienen ataques web de diferentes tipos como: *SQL injection, buffer overflow, information gathering, files disclosure, CRLF injection, XSS, server side include* y *parameter tampering*.

En el apartado 4.1.1 se describe detalladamente la base de datos, así como el tratamiento aplicado a esta base de datos.

2.4.2 UNSW-NB15

Se trata de una base de datos creada por la University of New South Wales [12] que contiene flujos de tráfico de red etiquetados como tráfico normal o tráfico anómalo.
Todo el tráfico está generado de forma artificial haciendo uso de la herramienta Ixia PerfectStorm [49] la cual hace uso de ataques conocidos para simular una serie de ataques y tráfico normal de red a tres servidores dispuestos para la generación de la base de datos. Todo el tráfico se almacena en formato pcap realizando un análisis del tráfico de red utilizando la herramienta *Tcpdump* [48]) y tras esto, es agrupado en flujos de tráfico (A.C) y de ellos se han extraído una serie de atributos. Estos atributos han sido obtenidos mediante el uso de diferentes herramientas (como Argus [47] y Bro-IDS [46]) y otros algoritmos de extracción de información.
Además de estar etiquetados como tráfico normal o tráfico anómalo, las entradas cuya clase es anomalía, disponen de otro atributo en el que se especifica qué tipo de ataque corresponde a ese flujo. Los diferentes tipos de ataque existentes son: *Exploits, Reconnaissance, DoS, Generic, Shellcode, Fuzzers, Worms, Backdoors* y *Analysis*.
En el apartado 4.1.2 y 4.1.1 se describe detalladamente la base de datos, así como el tratamiento aplicado a esta base de datos.

2.5 Complejidad de datos a clasificar

Uno de los grandes problemas cuando se afrontan problemas de este tipo, en el cual los conjuntos de datos son muy heterogéneos es la dificultad de poder generalizar la forma de clasificación para obtener resultados óptimos.
Es importante conocer la dificultad del problema de clasificación a la hora de elegir cómo se va a afrontar y una forma de observar la dificultad del problema es mediante el estudio de la complejidad de los datos. La mejor medida para estudiar esta complejidad es el error obtenido tras validar los resultados de un modelo, pero, sin embargo, este enfoque no es correcto para el enfoque de este trabajo porque se necesita un criterio lo más independiente posible de cualquier clasificador para obtener la complejidad de los datos de forma previa a la aplicación de ninguno.
Las dos medidas de la complejidad de los datos utilizadas en este trabajo son las siguientes [13][14]:

2.5.1 Proporción discriminante de Fisher

Se trata de un método que calcula cómo de separadas están dos clases en un atributo específico. El algoritmo para calcularlo es el siguiente:

$$\boldsymbol{FisherDR_f(X)} = \frac{\left(mean_{f+} - mean_{f-}\right)^2}{\sigma_{f+}^2 + \sigma_{f-}^2}, \qquad \textbf{(18)}$$

dónde:

- $\boldsymbol{f}$ es el atributo estudiado.
- $\boldsymbol{X}$ es el conjunto de datos
- $\boldsymbol{mean_{f+}}$ es la media para los elementos positivos (anomalías) en el atributo f en el conjunto de datos X.
- $\boldsymbol{mean_{f-}}$ es la media para los elementos negativos (normales) en el atributo f en el conjunto de datos X.

- $\boldsymbol{\sigma_{f+}^2}$ es la varianza media para los elementos positivos (anomalías) en el atributo f en el conjunto de datos X.
- $\boldsymbol{\sigma_{f-}^2}$ es la varianza para los elementos negativos (normales) en el atributo f en el conjunto de datos X.

Una vez tenemos las proporciones de cada uno de los atributos, se escoge la proporción máxima como proporción descriptora del conjunto de datos. Por lo tanto, la proporción discriminante de Fisher para un conjunto de datos completo es:

$$\boldsymbol{FisherDR(X)} = MAX_{f_i \in F}\left(FisherDR_{f_i}(X)\right), \quad \textbf{(19)}$$

dónde F es el conjunto de atributos del conjunto de datos X.
La idea de esta medida de complejidad es la de que si se encuentran elementos cuyo valor discriminante entre ambas clases sea muy alto, los datos serán más fácilmente separables y por lo tanto, la complejidad será menor.

2.5.2 Fracción de puntos en la frontera

Se trata de un método que calcula para cada punto del conjunto de datos sus k vecinos y comprueba cuántos de esos vecinos pertenecen a una clase contraria a la del punto estudiado. A estos puntos se les llama puntos en la frontera.
Finalmente, se calcula la fracción del total de puntos en la frontera para cada uno de los puntos del conjunto de datos entre el número total de puntos del conjunto de datos.
El algoritmo para calcular esta métrica es el siguiente:

$$\boldsymbol{FPF(X,k)} = \frac{\sum_{x \in X} |K_{c \neq c_x}|}{k * |X|}, \quad \textbf{(20)}$$

dónde:

- $\boldsymbol{X}$ es el conjunto de datos.
- **k** es el número de vecinos que se utilizarán en el algoritmo de K-Vecinos.
- **x** es cada uno de los puntos del conjunto de datos X.
- $\boldsymbol{|K_{c \neq c_x}|}$ es el número de vecinos de x con clase diferente a la de x.
- **|X|** es el número total de puntos del conjunto de datos de X.

Este método sigue la idea de que cuanto menor sea el valor de la fracción de puntos en la frontera, menor será la complejidad de los datos.

2.6 Ataques a Sistemas de Comunicaciones

Las bases de datos estudiadas contienen diferentes tipos de ataque que se realizan a través de los diferentes Sistemas de Comunicaciones expuestos para crear las mismas.
A continuación, se describen brevemente los ataques más comunes, presentes en las bases de datos estudiadas y que se dividen en los siguientes grandes grupos.

2.6.1 Ataques a aplicaciones web

Son ataques realizados contra aplicaciones web con el fin de obtener información de la aplicación o de sus usuarios o modificarla para obtener algún tipo de beneficio. Estos ataques se pueden realizar tanto del lado del cliente como de lado del servidor (estos últimos mucho más frecuentes).
Los ataques más frecuentes presentes en las bases de datos estudiadas son los siguientes:

- *SQL injection*: Ataque más común en aplicaciones web tal y como denomina la fundación OWASP [15]. Los atacantes intentan ejecutar sentencias SQL de forma fraudulenta en las bases de datos utilizadas por la aplicación web con el fin de obtener

información (como podrían ser los nombres de usuario y contraseñas del sitio web), modificar información (para, por ejemplo, hacer que cierto usuario sea administrador de la aplicación) o eliminar información con el fin de hacer que la aplicación web esté inoperativa. Este tipo de ataque se realiza a través de entradas mal comprobadas por parte de los usuarios o mala configuración en componentes que realizan consultas a la base de datos.

- *Buffer overflow.* Ataque que utiliza malas configuraciones o prácticas de programación en programas que utiliza la aplicación web con el fin de acceder a posiciones de memoria bien no reservadas o bien reservadas para otro uso para hacer inoperativa la aplicación u obtener información mediante esas posiciones de memoria [16].
- *Information gathering.* Ataque dedicado a obtener información de la aplicación web y de sus usuarios [17]. Comúnmente, este ataque se realiza previamente a otro tipo de ataque con el fin de conocer tanto el sistema como los usuarios y así poder realizar un ataque más enfocado y efectivo.
- *Files disclosure.* Ataque realizado con el fin de obtener ficheros de aplicaciones web a los cuales no deberíamos tener acceso haciendo uso de funciones vulnerables que permiten esta acción (en caso de que los desarrolladores hayan hecho uso de ellas) [18].
- *CRLF injection.* Si un servidor HTTP o HTTPS no procesa y filtra las peticiones recibidas correctamente, se podría inyectar código en las mismas a través del uso de CR (retorno de carro) y LF (salto de línea).
- *XSS (Cross Site Scripting).* Ataque que permite insertar código en aplicaciones web mediante el uso de elementos mal configurados o diseñados. Este ataque se encuentra en el número 7 del top 10 de OWASP [21] de ataques más comunes en aplicaciones web [20].
- *Server side include injection.* Ataque que hace uso de una utilidad (server side include) para crear páginas web de forma dinámica y que permite la ejecución de código en el lado del servidor [22].
- *Parameter tampering.* Ataque que se realiza manipulando valores de los parámetros enviados en las peticiones HTTP para intentar conseguir información de otros usuarios o elementos del sistema [23].

Estos ataques se relacionan con los ataques representados por la base de datos CSIC HTTP 2010.

2.6.2 Ataques a sistemas en red

Son ataques realizados directamente contra los sistemas como pueden ser ordenadores y máquinas virtuales que tienen servicios expuestos en red. Los más comunes en la base de datos UNSW-NB15 estudiada son:

- *Exploits.* Ataques que utilizan vulnerabilidades conocidas en programas o librerías para ejecutar código, obtener una Shell, robar información, etc [24].
- *Reconnaissance*. Ataques utilizados para obtener información de un sistema. Estos ataques se suelen realizar antes de realizar otro ataque con el fin de obtener información útil del sistema y descubrir potenciales vectores de ataque. Uno de los ataques de este tipo más conocidos es el escaneo de puertos [25].
- *DoS.* Ataque de denegación de servicio. Se realiza de diferentes maneras como explotando vulnerabilidades conocidas o realizándole un número de peticiones que no puede gestionar para que el sistema deje de estar operativo [26].
    - Otro tipo de ataque de denegación de servicio son los ataques DDoS (denegación de servicio distribuida) [26] dónde la denegación de servicio se realiza enviando peticiones al sistema desde una red distribuida para aumentar el número de estas.

3 Diseño

En este apartado se explica el diseño implementado para el desarrollo de este trabajo, así como la definición de los algoritmos utilizados en cada uno de los enfoques y la metodología a aplicar en cada uno de ellos junto con la definición de sus fases.

3.1 Medidas para evaluar los clasificadores de anomalías

A la hora de estudiar cada uno de los clasificadores que se van a implementar junto con las medidas que requiera cada uno, se van a utilizar una serie de métricas para comprobar cómo de bien detectan las anomalías. Teniendo en cuenta que la clase anomalía es tratada como la clase positiva y la clase normal es tratada como la clase negativa, para medir estos resultados, las métricas más importantes son la proporción de verdaderos positivos (TPR, también conocida como sensibilidad o *recall*) y la proporción de falsos negativos (FNR) que miden la cantidad de anomalías que detecta nuestro clasificador y la cantidad de anomalías que no detecta respectivamente.

En este sentido, cabe destacar que es más importante la métrica de la proporción de falsos negativos ya que al estar detectando anomalías en Sistemas de Comunicaciones, el error más crítico que se puede encontrar es el de dejar pasar una anomalía como un elemento normal dentro del sistema y de esta forma, permitir que un atacante pueda vulnerar nuestra seguridad.

Sin embargo, no solo son importantes estas métricas, sino que también es importante tener en cuenta que nuestro clasificador detecta bien las anomalías, pero no a costa de detectar muy mal el resto de los elementos normales.

Por esto, las métricas a tener en cuenta principalmente para el estudio de los clasificadores de anomalías son las métricas de recall (análoga a la proporción de detección) y el FPR ya que ambas métricas dan una visión global de un clasificador desde la perspectiva de detección de anomalías.

El recall muestra cuántas anomalías detecta nuestro clasificador del total de estas, el cual nos da una visión global tanto de cómo de bien detecta las anomalías, como del número de falsos negativos que se obtienen.

El FPR muestra cuántos elementos se están considerando como anomalía, cuando en realidad no lo son. Esta métrica si bien es menos vital que el recall debido a que es más grave un falso negativo que un falso positivo, es importante para no dejar pasar muchos elementos normales como anómalos.

Las métricas más comunes como son el *accuracy* o acierto o el área debajo de la curva ROC pierden valor en este tipo de casos ya que al estar ante conjuntos de datos tan desbalanceados, los datos obtenidos de estas métricas podrían aparentar ser muy buenos, ocultando que las anomalías se clasifican muy mal.

3.1.1 Métrica que maximiza la detección de anomalías y tráfico benigno

A lo largo de este trabajo, si bien se van a utilizar todas las métricas a modo orientativo, las métricas para tener en cuenta a la hora de comprobar el rendimiento de un clasificador en el contexto de anomalías de seguridad informática serían tanto el *recall* como el TNR ya que es primordial detectar el mayor número de anomalías posible, pero sin que ello conlleve pasar un gran número de elementos normales como anómalos ya que un gran número de alarmas falsas es también muy perjudicial. Con el fin de encontrar una métrica que maximice ambos valores para poder tener una visión global de cada uno de los clasificadores aplicados junto con las medidas pertinentes, se propone una métrica, *Maximal Recall and TNr*

(MRTN), que calcula la media harmónica entre ambos valores siguiendo el concepto de la *F-measure* de la siguiente manera:

$$\boldsymbol{MRTN} = 2 * \frac{recall * TNR}{recall + TNR}. \quad \textbf{(21)}$$

Esta métrica se propone para su uso a lo largo de este trabajo ya que las métricas *recall*, *TNR* y *F1Score* por separado no proporcionan el punto óptimo en todos los casos. Esta observación se ha realizado a lo largo de los experimentos realizados en este trabajo. A modo ilustrativo, y tal y como se observa en la Tabla 2, la cual muestra un conjunto de cuatro ejemplos de validación de un clasificador con diferentes hiperparámetros para un mismo conjunto de datos, el ejemplo número 1 sería el cual tiene la medida MRTN máxima y tal y como se observa, sus métricas recall, TNR y F1Score no son las máximas ya que estas corresponden a los ejemplos 2 (con recall máximo), 3 (con F1Score máximo) y 4 (con TNR máximo). Este ejemplo corresponde a la validación del clasificador *Local Outlier Factor* utilizando la base de datos CSIC HTTP 2010 y los hiperparámetros de cada uno de los ejemplos se pueden encontrar en el Anexo A.I.

Ejemplo	Recall	TNR	F1Score	MRTN
1	0.9433	0.8899	0.8357	**0.9158**
2	**0.9539**	0.8614	0.8121	0.9053
3	0.8455	0.9515	**0.8523**	0.8954
4	0.0438	**0.9998**	0.0839	0.0839

Tabla 2. Ejemplos de rendimiento MRTN.

Como se puede observar en la tabla, el ejemplo escogido utilizando la métrica MRTN es el ejemplo cuyos *recall* y TNR están maximizados en conjunto, ya que el ejemplo 2 muestra un *recall* más alto, pero un TNR más bajo lo cual hace que el promedio sea menor. Otro ejemplo que resaltar es el ejemplo 4 en el que se puede apreciar un caso muy extremo donde el TNR es prácticamente 1, pero su *recall* desciende prácticamente a 0.

3.2 Metodologías de operación

En este trabajo se van a aplicar una serie de procesos comunes y después una serie de procesos diferentes para la detección de anomalías para cada uno de los enfoques.
Los procesos comunes son los siguientes:

1. Estudio previo de la base de datos.
2. Extracción de atributos relevantes de la base de datos y creación de una nueva base de datos con estos atributos.
3. Estudio de la complejidad de los datos.

Los procesos siguientes para cada enfoque son los siguientes:

- Enfoque clasificación multiclase:

4. Reducción de la clase anomalía hasta valores cercanos al 0% de presencia. En este punto se realizan varias reducciones para así estudiar el efecto en los diferentes porcentajes de presencia de anomalía.
5. Aplicación de métodos de *undersampling* y *oversampling* a los datos para cada uno de los conjuntos de datos con diferente desbalanceo de la fase anterior.
6. Estudio de las métricas con los diferentes clasificadores multiclase e hiperparámetros de cada clasificador para cada uno de los conjuntos de datos con diferente desbalanceo de la fase anterior.

- Enfoque de clasificación de clase única

4. Estudio de las métricas con los diferentes clasificadores *OneClassClassification* e hiperparámetros de cada clasificador.

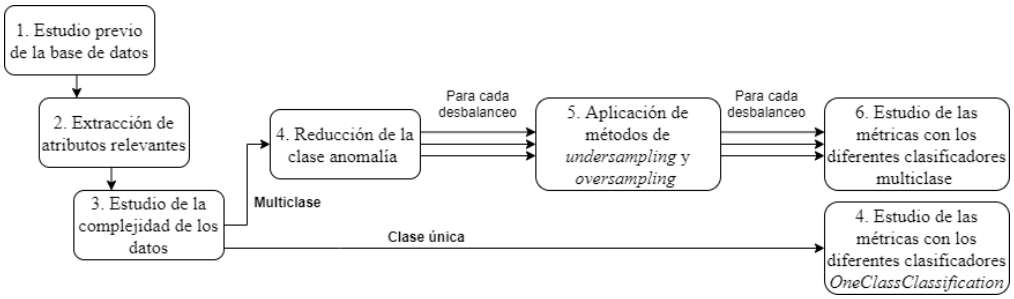

Figura 3. Diagrama de flujo del experimento.

Cada una de las fases se definen a continuación y se detallarán sobre las bases de datos en el apartado 4.

3.2.1 Estudio previo de la base de datos

Con cada una de las bases de datos que se utilizarán en este trabajo, primero se realizará un estudio manual apoyado de la documentación o artículo oficial de cada una de ellas con el fin de obtener información sobre el contenido de la base de datos, así como una visión global sobre la misma y sus atributos.

3.2.2 Extracción de atributos relevantes

En este apartado, es importante tener en cuenta el contexto en el que nos encontramos ya que la mayoría de las bases de datos al estar relacionadas con eventos de seguridad en Sistemas de Comunicaciones, contienen atributos representativos de las personas que han realizado los eventos tales como IPs fuente y destino, puertos fuente y destino, etc. que son elementos que si bien para un sistema de protección como podría ser un antivirus, un IDS, un *firewall* o un *WAF* pueden ser muy útiles para aplicar técnicas de *whitelisting* y *blacklisting* tanto de IPs como de puertos, a la hora de crear y entrenar clasificadores para la detección de anomalías no son útiles ya que aportarían un sesgo muy importante al clasificador y dejaría de comportarse adecuadamente en eventos generados por nuevos posibles atacantes (con una IP distinta) o por nuevos métodos de ataque (que harían variar el puerto destino por ejemplo) y por eso serán eliminados siempre en esta fase.
En este proceso se buscará encontrar los elementos más discriminantes con el fin de obtener el conjunto de datos más reducido posible sin pérdida de información importante a la hora de clasificar cada elemento.

3.2.3 Estudio de la complejidad de los datos

Tras realizar una extracción de los atributos relevantes y haber obtenido un conjunto de datos final con el que se va a trabajar, se aplicará un proceso de estudio de la complejidad de los datos haciendo uso de los algoritmos explicados previamente para disponer de unas métricas con las que poder medir tras la validación la relevancia de la complejidad de los diferentes conjuntos de datos en el proceso de detección de anomalías.
Este estudio sirve a modo indicativo de la dificultad de cada uno de los problemas a la hora de clasificar, y en el apartado de conclusiones se comenta su influencia en la detección de anomalías.

3.2.4 Reducción de la clase anomalía

En todas las bases de datos estudiadas el porcentaje de anomalías está por debajo del porcentaje de elementos normales, pero al estar generadas mediante generadores sintéticos, son porcentajes muy por encima de valores que representarían la realidad en la que las anomalías representan un porcentaje mínimo del total de los elementos. Por esto, en este

proceso se eliminarán de forma aleatoria los elementos anómalos hasta llegar a valores cercanos al 0% para simular un entorno más real a la hora de entrenar y validar los diferentes clasificadores multiclase (en los clasificadores *OneClassClassification* este proceso no tendría sentido ya que en la fase de entrenamiento solo se utilizan los elementos normales y el porcentaje de elementos anómalos es indiferente). Para tener un estudio más completo, se tendrán en cuenta una serie de puntos con diferentes porcentajes de representación de la clase anomalía en todo el conjunto de la base de datos para comprobar el funcionamiento de los métodos de detección de anomalías estudiados en cada uno de ellos y disponer de unos resultados con más perspectiva para cada desbalanceo de clases.
Para realizar esta reducción se aplica el algoritmo *Random Undersampling* para reducir la clase anomalía de forma aleatoria y además se aplica varias veces promediando los resultados para obtener resultados que no dependan de la reducción aplicada.

3.2.5 Aplicación de métodos de *oversampling* y *undersampling*

En este apartado, se aplicarán los diferentes métodos de *undersampling* y *oversampling* detallados previamente de forma separada para equiparar los porcentajes de elementos normales y anómalos.
En el caso de métodos de *undersampling* se reducirá la clase normal y en el caso de métodos de *oversampling* se aumentarán los ejemplos anómalos para, de esta forma, equiparar los porcentajes y estudiar si se mejoran los resultados de detección de anomalías aplicando estos métodos en cada uno de los diferentes porcentajes de presencia de anomalía.

3.2.6 Estudio de las métricas con los diferentes clasificadores multiclase

Una vez se han realizado los procesos anteriores, para cada uno de los clasificadores multiclase y cada uno de los posibles hiperparámetros que se estudiarán de cada uno, se realizará una validación cruzada con *K-folds* (con $K = 5$ para que el tamaño del conjunto de prueba sea del 20%) y varias repeticiones y después promediando, para obtener todas las métricas descritas previamente.
Tras aplicar estudios de búsqueda de hiperparámetros con una búsqueda en cuadrícula para los diferentes clasificadores multiclase, en las bases de datos se decide utilizar los siguientes clasificadores e hiperparámetros correspondientes a la librería scikit-learn de Python [45]:

- Árboles de decisión: DecisionTreeClassifier(criterion='entropy').
- Random Forest: RandomForestClassifier(criterion='entropy').
- Vecinos próximos: kNeighborsClassifier(n_neighbors=5).

3.2.7 Estudio de las métricas con los diferentes clasificadores *OneClassClassification*

Una vez se han realizado los procesos anteriores, para cada uno de los clasificadores *OneClassClassification* y cada uno de los posibles hiperparámetros que se estudiarán de cada uno, se realizará una separación en conjuntos de test-train con un tamaño de test del 20% del total, después del conjunto de train se eliminarán los elementos anómalos para reducirlo a un espacio de una única clase y así poder entrenar clasificadores de clase única y validar con un espacio de datos tanto normales como anómalos (el 20% de los datos extraídos en la fase de separación en los conjuntos de train-test). Este proceso de validación se aplicará varias veces y se promediará para obtener resultados más específicos y no depender de una única separación del conjunto de datos.
Tras aplicar estudios con los clasificadores de clase única en las bases de datos se decide utilizar los siguientes clasificadores e hiperparámetros correspondientes a la librería scikit-learn de Python [40]:

- **EllipticEnvelope**(assume_centered=True, support_fraction=0.5)
- **IsolationForest**(bootstrap=True, max_features=5,n_estimators=20).
- **LocalOutlierFactor**(leaf_size=10, metric='braycurtis', n_neighbors=3).

Y el **autoencoder** utilizado será un autoencoder con cuatro capas ocultas de tamaño $N, \frac{N}{2}, \frac{N}{2}, N$ con $N = número\ de\ atributos$ y con funciones de activación tangente hiperbólica, RELu, RELu, tangente hiperbólica, respectivamente.

4 Desarrollo, pruebas y resultados

4.1 Preparación y estudio inicial de las bases de datos

Las tres primeras fases definidas en la Figura 2 para este trabajo son comunes a ambos enfoques, por lo que se exponen de forma general para ambas fases.
En este apartado se muestra el desarrollo de las fases de estudio previo, extracción de atributos relevantes y estudio de la complejidad de los datos.

4.1.1 Base de datos CSIC HTTP 2010

4.1.1.1 Estudio previo de la base de datos

Esta base de datos creada por Carmen Torrano [1] se compone de una serie de peticiones HTTP etiquetadas como tráfico normal o anómalo. En el Anexo A.B se indica una descripción más detallada del funcionamiento de las peticiones HTTP.
El número total de elementos es 97.065, de los cuales 72.000 corresponden a peticiones normales y 25.065 corresponden a peticiones anómalas. Estos valores representan un 74,18% de peticiones normales y un 25,82% de peticiones anómalas. En el Anexo A.A.1(1) se muestra un ejemplo en formato plano de estas peticiones y en el Anexo A.B se definen las peticiones HTTP de forma general.
De estas peticiones se pueden obtener los siguientes atributos: *Method, URL, Protocol, User-Agent, Pragma, Cache-control, Accept, Accept-Encoding, Accept-Charset, Accept-Language, Host, Cookie, Content-Type, Connection, Content-Length, Payload, Label.*
Otra forma de obtener información de estos datos es mediante la extracción de los 1-gramas en los caracteres del cuerpo de cada una de las peticiones (o parámetros en la ruta en caso de ser una petición GET).
Los n-gramas se obtienen analizando el número de apariciones de cadenas de caracteres de tamaño n en un texto, y en este caso en particular, al tratarse de 1-gramas se obtiene el número de apariciones de cada carácter ASCII en el cuerpo de la petición (o parámetros en la ruta) obteniendo así 256 atributos más (uno por cada carácter ASCII).

4.1.1.2 Extracción de atributos relevantes

Una vez obtenidos todos estos atributos del conjunto de datos inicial, y con el fin de obtener un conjunto de datos de referencia con el cual poder comparar los resultados obtenidos en este trabajo, se ha utilizado el enfoque tomado por Carmen Torrano en su tesis [11] en el cual se extraen una serie de atributos en base a los atributos presentes en el conjunto de datos junto con una serie de 1-gramas para formar un conjunto de datos que favorezca la clasificación de las anomalías en función a la aplicación. En la tesis se utilizan varios métodos de selección de atributos, y en este trabajo se ha utilizado el método definido en la tesis cómo *combine-select* en el cual, el conjunto de datos final se obtiene combinando una serie de atributos obtenidos mediante "conocimiento experto" tras procesar la base de datos y el conjunto de datos de 1-gramas, y aplicando selección de atributos a en el subconjunto mediante el algoritmo *mRMR* (*minimum Redundancy Maximum Relevancy*) el cual se encarga de seleccionar los atributos que aporten la mayor relevancia posible y la menor redundancia posible. De este método se obtiene un conjunto de datos con los siguientes atributos:

- **req_lengths**: Longitud total de la petición.
- **path_lenghts**: Longitud total de la ruta solicitada en la petición.
- **arg_lenghts**: Longitud total de los argumentos de la petición.
- **arg_letters_num**: Número de letras en los argumentos de la petición.

- **arg_digits_num**: Número de dígitos en los argumentos de la petición.
- **arg_special_num**: Número de caracteres especiales en los argumentos de la petición (se entiende como carácter especial cualquiera no alfanumérico).
- **path_letter_num**: Número de letras en el path solicitado en la petición.
- **distinct_bytes**: Número de bytes distintos en la petición.
- **d**: número de "d" en los argumentos de la petición.
- **3**: número de "3" en los argumentos de la petición.
- **p**: número de "p" en los argumentos de la petición.
- **-**: número de "-" en los argumentos de la petición.
- **i**: número de "i" en los argumentos de la petición.
- **t**: número de "t" en los argumentos de la petición.
- **P**: número de "P" en los argumentos de la petición.
- **Label**: etiqueta de la petición con dos valores posibles, "normal" o "anomalía".

Nota: al conjunto de atributos seleccionado, le falta el atributo "número de *keywords*" que se utiliza en el trabajo [11] .Esto se debe a que, si bien se hace referencia en el trabajo a este conjunto de palabras clave, no se han encontrado en ningún sitio y por lo tanto se ha decidido descartar su uso para este trabajo.

La tabla descriptiva del conjunto de datos final es la siguiente:

	req_lengths	path_lenghts	arg_lenghts	arg_letters_num	arg_digits_num
Media	445,4248	63,18841	27,64981	19,11377	4,379004
Desviación típica	97,20113	70,22281	70,20025	47,14428	16,19092
Valor mínimo	327	8	0	0	0
Valor máximo	1252	872	836	312	437

	arg_special_num	path_letter_num	distinct_bytes	d	3
Media	4,169227	46,1621	64,33434	1,032895	0,46644
Desviación típica	10,6685	47,13349	3,003931	2,645985	1,597342
Valor mínimo	0	4	57	0	0
Valor máximo	152	339	74	18	34

	P	-	i	t	P
Media	0,501087	0,064091	1,701004	0,7189	0,078102
Desviación típica	1,361259	0,379705	4,360355	1,765709	0,346696
Valor mínimo	0	0	0	0	0
Valor máximo	11	7	32	20	5

Tabla 3. Descripción del conjunto de datos final CSIC HTTP 2010.

En la tabla podemos observar, para cada uno de los atributos del conjunto de datos, su media, desviación típica, valor máximo y valor mínimo del total de elementos en el conjunto de datos.

En el Anexo A.A.1(2) se muestra un ejemplo del conjunto de datos tras la fase de procesamiento.

4.1.1.3 Estudio de la complejidad de los datos

Tras estudiar la complejidad de los datos con las métricas comentadas en el apartado 2.5, se han obtenido los siguientes valores presentes en la tabla.

Discriminante de Fisher (máximo de los discriminantes de todos los atributos)	Fracción de puntos en la frontera
0.2959 (atributo distinct_bytes)	4.2997%

Tabla 4. Complejidad de los datos CSIC HTTP 2010

Se puede observar que el atributo más discriminante según el discriminante de Fisher es distinct_bytes el cual muestra el número de bytes distintos existentes en la petición.
Al no tener una referencia de como de complejo es el conjunto de datos siguiendo el criterio de la discriminante de Fisher, se muestra en la Figura 4 la distribución por clases del atributo más relevante (distinct_bytes) con el fin de observar cómo de solapadas están ambas clases.

4.1.2 Base de datos UNSW-NB15

4.1.2.1 Estudio previo de la base de datos

Esta base de datos creada por creada por la University of New South Wales [12] que contiene flujos de tráfico de red etiquetados como tráfico normal o tráfico anómalo.
El número total de elementos es 2.540.047, de los cuales 2.218.764 corresponden a flujos normales y 321.283 corresponden a flujos anómalos. Estos valores representan un 87.35% de flujos normales y un 12.65% de flujos anómalos. Se trata de una base de datos alrededor de 25 veces mayor que la base de datos CSIC HTTP 2010.
De estos flujos se pueden obtener los siguientes atributos: *srcip, sport, dstip, dsport, proto, state, dur, sbytes, dbytes, sttl, dttl, sloss, dloss, service, Sload, Dload, Spkts, Dpkts, swin, dwin, stcpb, dtcpb, smeansz, dmeansz, trans_depth, res_bdy_len, Sjit, Djit, Stime, Ltime, Sintpkt, Dintpkt, tcprtt, synack, ackdat, is_sm_ips_ports, ct_state_ttl, ct_flw_http_mthd, is_ftp_login, ct_ftp_cmd, ct_srv_src, ct_srv_dst, ct_dst_ltm, ct_src_ ltm, ct_src_dport_ltm, ct_dst_sport_ltm, ct_dst_src_ltm, attack_cat* y *Label*.

4.1.2.2 Extracción de atributos relevantes

En este caso, se han eliminado primero los elementos relacionados con IPs, puertos, tiempos y los particulares para un único protocolo ya que tal y como se explica en el apartado 0.2, estos atributos no aportan valor en este estudio al utilizar bases de datos generadas artificialmente y en entornos muy cerrados.
Tras esto, y al disponer de un conjunto muy diverso de atributos, se ha entrenado un árbol de clasificación y se han escogido todos los atributos que tienen una importancia mayor al 0.4%. Esta selección se aplica asumiendo que elegir estos atributos cubre casi el total de la importancia de Gini normalizada para el conjunto de datos. Tras realizar la selección que se muestra en el Anexo A.G, se observa que se cubre el 95.499% del total de importancia de Gini para los atributos por lo que se asume que el conjunto de datos se explica casi en su totalidad con esta selección de atributos. Para comprobar que dicho árbol de decisión es adecuado para la selección de parámetros se ha validado dicho árbol de decisión con el conjunto de datos completo y se ha obtenido un 0.9927 de *accuracy*, un 0.9710 de *recall*, un 0.9958 de TNR y un 0.9832 de MRTN.

Finalmente, la selección de atributos (ordenados según su importancia) es la siguiente:

- **sttl**: Valor del "*Time to live"* recibido. Se refiere el número de saltos máximo permitido para un paquete antes de ser descartado.
- **Sload**: Bits por segundo recibidos.
- **ct_srv_dst**: Número de conexiones que tienen el mismo servicio y la misma fuente en las últimas 100 conexiones.
- **sbytes:** número de bytes recibidos.
- **smeansz:** media del tamaño de paquetes recibidos.
- **Spkts:** número de paquetes recibidos.
- **synack**: Tiempo de conexión TCP.
- **Label**: Etiqueta del elemento con los valores "normal" (0) o "anomalía" (1).

La tabla descriptiva del conjunto de datos final es la siguiente.

	sbytes	sttl	Sload	Spkts	smeansz
Media	4339,6	62,78197	36956448	33,28884	124,2536
Desviación típica	56405,99	74,62277	1,19E+08	76,28388	151,9162
Valor mínimo	0	0	0	0	0
Valor máximo	14355774	255	5,99E+09	10646	1504

	synack	ct_srv_dst	label
Media	0,003288	8,988958	0,126487
Desviación típica	0,025936	10,82249	0,332398
Valor mínimo	0	1	0
Valor máximo	4,525272	67	1

Tabla 5. Descripción del conjunto de datos final UNSW-NB15.

En la tabla podemos observar, para cada uno de los atributos del conjunto de datos, su media, desviación típica, valor máximo y valor mínimo del total de elementos en el conjunto de datos.

En el Anexo A.A.2(2) y A.A.1(2) se muestra un ejemplo del conjunto de datos tras la fase de procesamiento.

Observando las importancias de cada atributo (Anexo A.G) se observa que el atributo sttl tiene una importancia muy alta y observando las medias de los valores de este atributo para elementos anómalos y normales se aprecia como los elementos anómalos tienen una media cercana a 255 (valor máximo permitido de este atributo) y como los elementos normales tienen una media cercana a 4 con lo cual surge la pregunta de si esto es debido a la generación artificial de los datos dónde se asignan valores muy altos para este atributo en los ataques simulados o si realmente los ataques tienen un patrón de utilizar valores de sttl altos como por ejemplo, en ataques que pasen por muchos nodos para camuflar el rastro del atacante.

4.1.2.3 Estudio de la complejidad de los datos

Tras estudiar la complejidad de los datos con las métricas comentadas en el apartado 2.5, se han obtenido los siguientes valores presentes en la tabla.

Discriminante de Fisher (máximo de los discriminantes de todos los atributos)	Fracción de puntos en la frontera
12.5354 (atributo sttl)	1.343%

Tabla 6. Complejidad de los datos UNSW-NB15

Se puede observar que el atributo más discriminante es sttl lo cual tiene sentido por el mismo motivo que se ha comentado previamente.
Para disponer de una visión de cómo de discriminante es este atributo se puede observar en la Figura 5 la distribución por clases del atributo dónde se observa claramente como ambas distribuciones están claramente separadas.

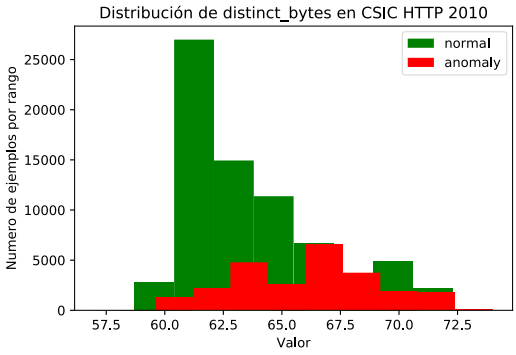

Figura 4. Distribución del atributo más discriminante (Fisher) de CSIC HTP 2010.

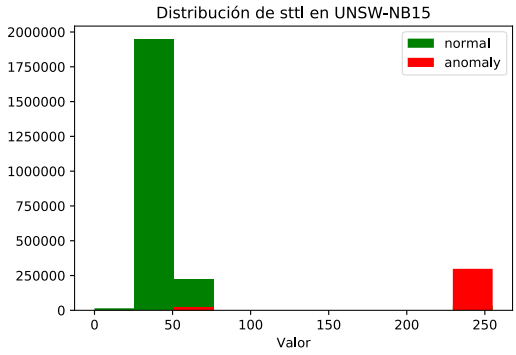

Figura 5. Distribución del atributo más discriminante (Fisher) de UNSW-NB15.

4.2 Enfoque multiclase con métodos de oversampling y undersampling

Una vez finalizadas las fases comunes definidas en el apartado 0 y obtenidos los conjuntos de datos finales con los que trabajar, se procede a realizar las fases correspondientes al enfoque de clasificación multiclase que comprende las fases de Reducción de la clase anomalía, Aplicación de métodos de *oversampling* y *undersampling* y estudio de las métricas con los diferentes clasificadores multiclase. Con este enfoque se estudia la importancia de aplicar estos métodos cuando existe un desbalanceo de clases muy notable y además se estudia para diferentes desbalances para estudiar la importancia para cada uno de ellos.

4.2.1 Reducción de la clase anomalía

Los valores de representación de la clase anomalía en las bases de datos estudiadas, originalmente, son los correspondientes a la siguiente tabla.

CSIC HTTP 2010	UNSW-NB15
25.82%	12.65%

Tabla 7. Porcentaje de anomalías en las bases de datos.

Teniendo en cuenta estos valores, la reducción de la clase anomalía que se aplica en cada una de las bases de datos para realizar el estudio del desbalanceo de clases para diferentes desbalances son los siguientes correspondientes con los valores $N, \frac{N}{2}, \frac{N}{4}, \frac{N}{8}$ y 1% dónde N es el porcentaje inicial de anomalías en cada base de datos:

CSIC HTTP 2010	UNSW-NB15
25.82%, 12.91%, 6,45%, 3.22%, 1%	12.65%, 6.32%, 3.16%, 1.58%, 1%

Tabla 8. Porcentajes de anomalías estudiados para cada base de datos.

Tras reducir la clase anomalía en cada una de las bases de datos, los números de ejemplos de la clase anomalía para cada una de las bases de datos con los porcentajes correspondientes son los correspondientes a la siguiente tabla.

CSIC HTTP 2010 (elementos normales: 72000)	UNSW-NB15 (elementos normales: 2218764)
25065, 10674, 4968, 2401, 727	321283, 149796, 72452, 35644, 22411

Tabla 9. Número de elementos anómalos estudiados para cada base de datos.

Una vez reducidos los ejemplos de anomalías de cada una de las bases de datos, se estudian las métricas principales (*Recall*, TNR y MRTN) en cada una de ellas para observar las diferencias en las mismas con respecto al decrecimiento de la clase anomalía.

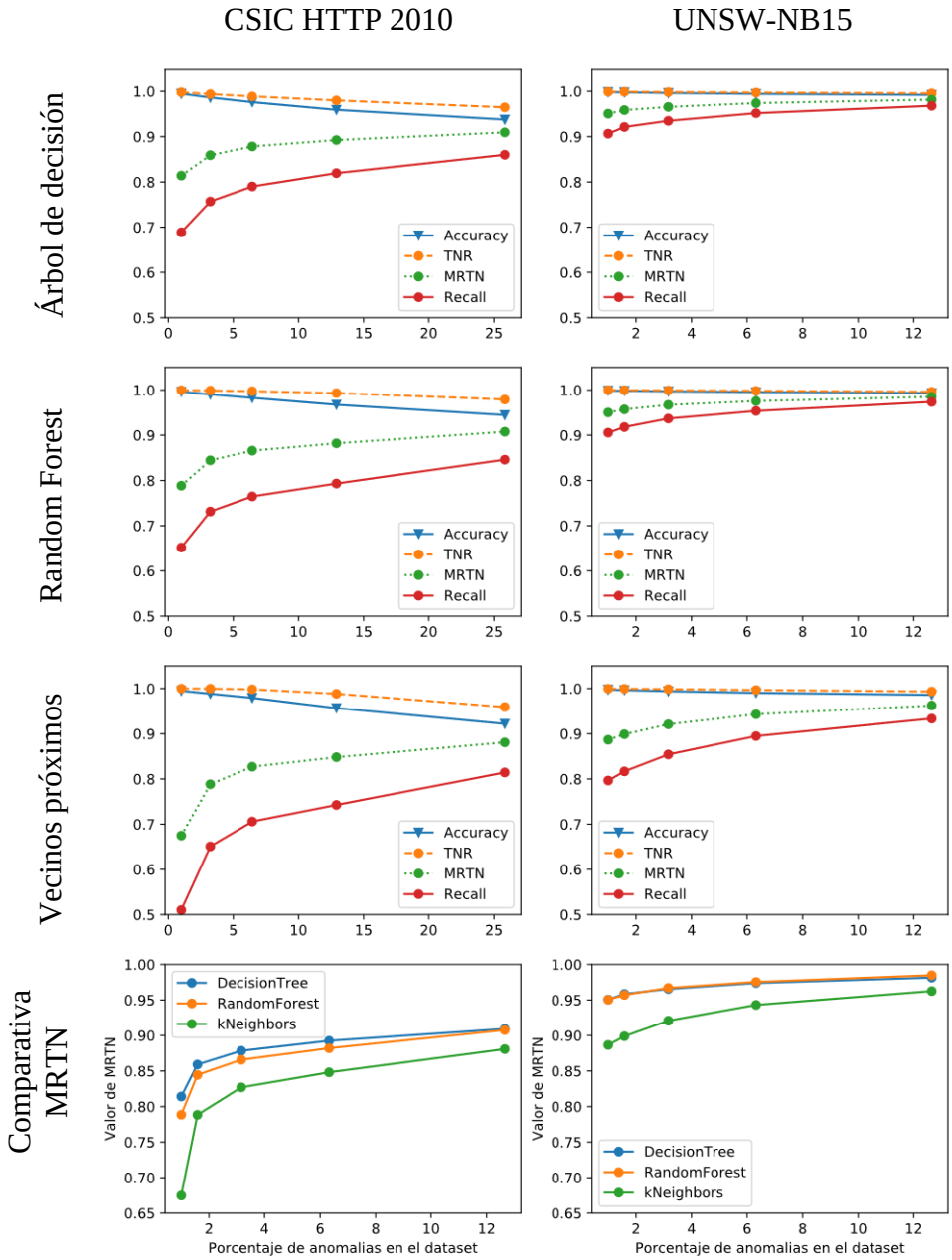

Figura 6. Comparativa de métricas para cada clasificador reduciendo la clase anomalía.

Se puede apreciar en las gráficas que, al reducirse el porcentaje de anomalías en las bases de datos, el valor de TNR aumenta hasta llegar casi a 1 ya que al ir reduciendo la clase anomalía se va quedando la clase mayoritaria casi sola en los conjuntos de datos, y, por lo tanto, el modelo se ajusta a la clase mayoritaria. Este efecto también ocurre en el caso del *accuracy* y por esto se comprueba que no es una medida que represente el rendimiento de un clasificador cuando estamos en conjuntos de datos con la problemática del desbalanceo de clases.
Sin embargo, también se puede observar que, al ir reduciendo el porcentaje de anomalías en la base de datos el valor de recall comienza a caer de forma considerable y tal y cómo se observa por el valor de MRTN, la pérdida de *recall* es mayor que la ganancia de TNR por lo que se puede concluir que las anomalías se clasifican peor cuanto menor es su presencia en el conjunto de datos.
Comparando las gráficas de ambas bases de datos, principalmente en las figuras de la última fila de la Figura 6, se observa que los resultados generales en la detección de anomalías de la base de datos UNSW-NB15 son mejores que los resultados generales de la base de datos CSIC HTTP 2010, lo cual concuerda con las medidas obtenidas de complejidad de los datos dónde se aprecia que la complejidad de la base de datos UNSW-NB15 es mucho menor que la complejidad de CSIC HTTP 2010.
Observando ambas bases de datos también comprobamos que los clasificadores que mejores resultados obtienen son los clasificadores de árboles de decisión y *Random Forest* siendo *Random Forest* el que mejores resultados obtiene (aunque muy cerca de los resultados de árboles de decisión).
Si observamos ambas bases de datos, y con el fin de poder compararlas en el mismo grado de desbalanceo de clases ya que estamos estudiando el problema de detección de anomalías dónde este grado es muy importante, se toman como referencia el primer punto de las gráficas de la base de datos UNSW-NB15 y el segundo punto de las gráficas de la base de datos CSIC HTTP 2010 dónde ambas tienen una presencia de anomalías cercana al 12%. En estos puntos se observa que, aunque ambas bases de datos sean completamente diferentes (flujos de tráfico de red, a nivel de red, frente a peticiones HTTP, a nivel de aplicación), se aprecia que *Random Forest* es el que mejor funciona en ambas (a nivel de MRTN).

4.2.2 Aplicación de métodos de *oversampling* y *undersampling* y estudio de las métricas con los diferentes clasificadores multiclase

A continuación, se exponen los resultados de aplicar los mismos clasificadores, pero tras aplicar técnicas de *oversampling* y *undersampling* en cada uno de los puntos de desbalanceo de clases para observar si el rendimiento mejora tras aplicarlas.
Para cada uno de los métodos se muestra una gráfica con la ganancia (G) de MRTN que aporta aplicar el método aplicado dónde:

$$G(X) = (MRTN(X)_{(over/under)sampled} - MRTN(X)_{normal}) * 100 \quad \textbf{(22)}$$

Dónde:

- $MRTN(X)_{(over/under)sampled}$ indica el valor de MRTN tras validar el modelo dividiendo los datos X en dos conjuntos de entrenamiento y test y sub/sobre muestreando los datos para el entrenamiento, es decir, utilizando datos sintéticos para el entrenamiento del clasificador.
- $MRTN(X)_{normal}$ indica el valor de MRTN tras validar el modelo dividiendo los datos X en dos conjuntos de entrenamiento y test.

La ganancia muestra cuánto mejora o empeora aplicar el método al conjunto de datos dónde si $G > 0$ observamos que aplicar ese método mejora el rendimiento y si $G < 0$ observamos que aplicar ese método empeora el rendimiento.
Este análisis solo pretende mostrar si la aplicación de los algoritmos de oversampling y undersampling tienen una mejora en el rendimiento de la detección de anomalías. Estos datos se fusionan junto con los detalles numéricos presentes en el Anexo mostrando los resultados finales en la sección 4.5.
En las figuras a continuación, los ejes Y están en una escala diferente para que se puedan apreciar correctamente las ganancias de cada clasificador y algoritmo de *oversampling* o *undersampling.*

4.2.2.1 Métodos de oversampling

Los métodos de oversampling que se han considerado son los métodos de *SMOTE*, *Boderline SMOTE1* y *ADASYN*.
Utilizando cada uno de los puntos de cada base de datos, se ha estudiado si aplicar estos algoritmos de *oversampling* a la clase minoritaria tiene un efecto positivo en la clasificación. Para medir este efecto, se ha comprobado la diferencia entre las diferentes métricas con *oversampling* y sin él para observar si se mejoran las mismas aplicando *oversampling* o no. La Figura 7 a continuación se muestran los diferentes valores para la diferencia con la métrica MRTN para los diferentes clasificadores utilizados.

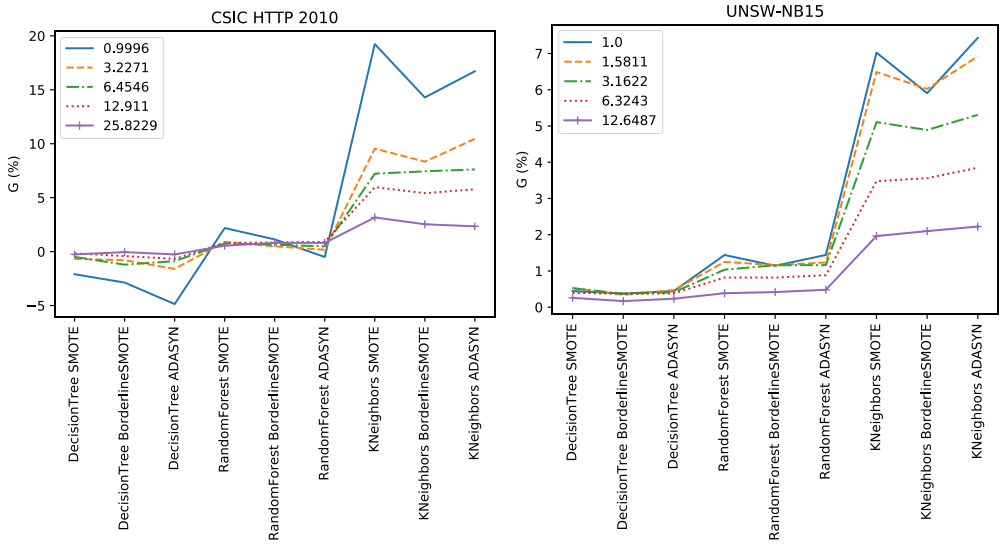

Figura 7. Ganancia en diferentes desbalances de clase aplicando algoritmos de oversampling.

Cómo se puede observar, los valores de MRTN aplicando oversampling son muy similares a los valores sin aplicar (diferencias menores de 0.01) para árboles de decisión independientemente del algoritmo utilizado. Con el algoritmo Random Forest también se observan valores ligeramente superiores, pero sin llegar a poder considerarse una mejora significativa. Con el algoritmo de vecinos próximos sí que se aprecian mejoras considerables y proporcionales inversamente al porcentaje de anomalías en el conjunto de datos, esto quiere decir, que cuanto menor es el número de anomalías en el conjunto de datos, mayor es la mejora que aportan los algoritmos de *oversampling*.
En ambas bases de datos se comporta de forma similar y además se observa que los algoritmos de *oversampling* funcionan de forma parecida en todos los casos a excepción de los casos con anomalías muy reducidas en los cuales se pueden ver variaciones mayores.

Tras haber comprobado las diferencias que provocan los algoritmos de *oversampling*, se observan las tablas del Anexo A.D con las diferentes métricas estudiadas (*Recall*, TNR y MRTN) para cada uno de los diferentes porcentajes de anomalías en el conjunto de datos y para cada una de las bases de datos para observar qué combinación de algoritmos funciona mejor en cada caso.
En dichas tablas podemos observar que para todas las bases de datos el algoritmo de vecinos próximos con la clase anomalía reducida no es el mejor, pero aplicándole la técnica de *oversampling* su rendimiento mejora considerablemente, convirtiéndolo en el algoritmo con el mejor MRTN en la base de datos CSIC HTTP 2010 y con valores de MRTN cercanos al mejor valor en la base de datos UNSW-NB15; pero para los algoritmos de clasificación de árboles de decisión y *Random Forest* la mejora que aportan los algoritmos de *oversampling* en ambas bases de datos no suponen un incremento considerable en términos de rendimiento medido con MRTN.

4.2.2.2 Métodos de undersampling

Los métodos de undersampling que se han considerado son *RandomUndersampler* y *Near Miss*.
Utilizando cada uno de los puntos de cada base de datos, se ha estudiado si aplicar estos algoritmos de *undersampling* a la clase mayoritaria tiene un efecto positivo en la clasificación. Para medir este efecto, se ha comprobado, al igual que en el apartado anterior, la diferencia entre las diferentes métricas con *undersampling* y sin él para cada número de puntos total dónde el número de anomalías es igual que el número de elementos normales.
La Figura 8 a continuación muestra los diferentes valores para la diferencia con la métrica MRTN para los diferentes clasificadores utilizados.

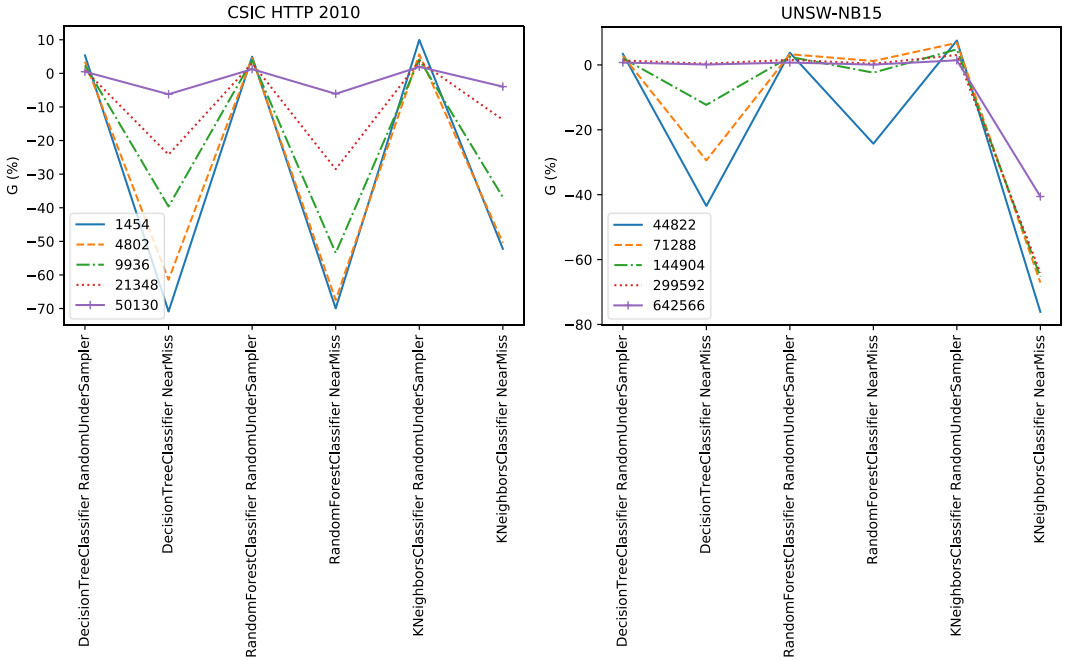

Figura 8. Ganancia en diferentes desbalances de clase aplicando algoritmos de undersampling.

Cómo se puede observar, aplicar métodos de *undersampling* hace que el rendimiento (medido con MRTN) empeore considerablemente al aplicar *Near Miss* y se observan ciertas mejoras poco notables al aplicar *Random Oversampler*.

Tras haber comprobado las diferencias que provocan los algoritmos de *undersampling*, se observan las tablas del A.E con las diferentes métricas estudiadas (*Recall*, TNR y MRTN) para cada uno de los diferentes números totales de puntos en el conjunto de datos y para cada una de las bases de datos para observar qué combinación de algoritmos funciona mejor en cada caso. En la sección 4.5 se puede observar la fusión de estos resultados con el rendimiento de los clasificadores con los resultados finales.
Tal y cómo podemos observar en las tablas, se comprueba que aplicar el algoritmo *Near Miss* afecta notablemente de forma negativa a la clasificación de anomalías, y aplicar *RandomOversampling* mejora, aunque en proporciones poco notables, el valor de MRTN para todos los algoritmos. También se observa que los mejores algoritmos son los de árbol de decisión y *Random Forest.*

4.2.2.3 Métodos de oversampling y undersampling conjuntos

Los algoritmos aplicados son los de *SMOTE + ENN* y *SMOTE + Tomek Links*.
Utilizando cada uno de los porcentajes de presencia de anomalías en cada una de las bases de datos se ha probado que aplicar algoritmos de *oversampling* y *undersampling* conjuntamente para equiparar la clase anomalía y la clase normal tiene un efecto positivo en el rendimiento de los clasificadores (medido con MRTN). La Figura 9 a continuación muestra la diferencia de MRTN aplicando los algoritmos menos MRTN sin aplicarlos para cada una de las bases de datos estudiadas.

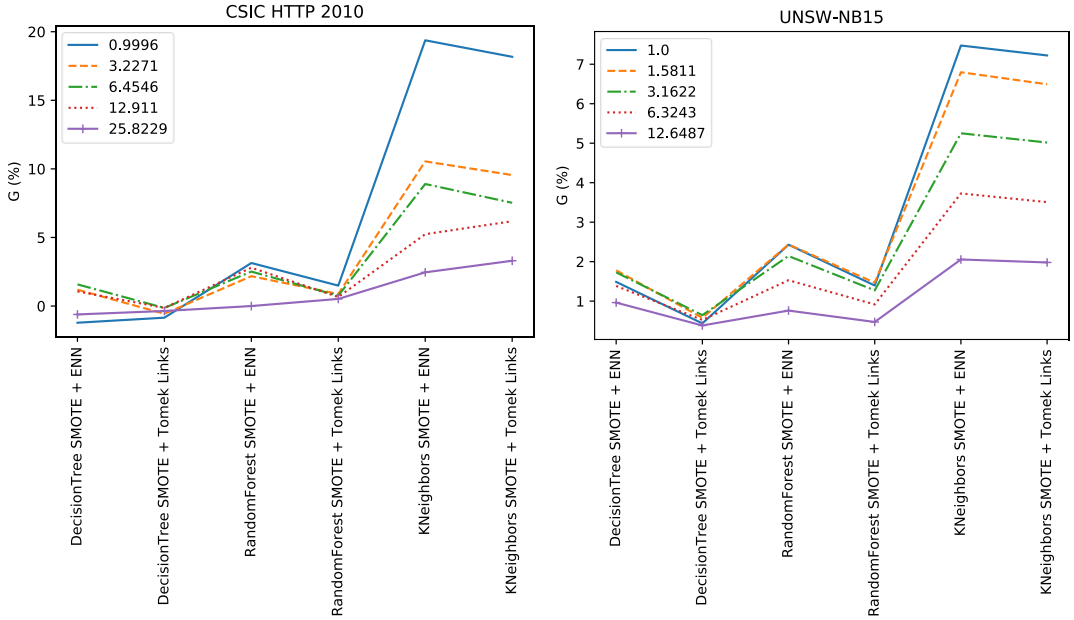

Figura 9. Ganancia en diferentes desbalances de clase aplicando algoritmos de oversampling y undersampling conjuntos.

Cómo se puede observar, aplicar métodos de *undersampling* y *oversampling* conjuntamente utilizando los algoritmos descritos en la sección 2.3.1.1 hace que el rendimiento (medido con MRTN) mejore en casi todos los casos.
Tras haber comprobado las diferencias que provocan los algoritmos de *undersampling*, se observan las tablas del Anexo A.F con las diferentes métricas estudiadas (*Recall*, TNR y MRTN) para cada uno de los diferentes números totales de puntos en el conjunto de datos y para cada una de las bases de datos para observar qué combinación de algoritmos funciona mejor en cada caso.
Se puede observar en las tablas del Anexo A.F, que para la base de datos CSIC HTTP 2010, el algoritmo con más rendimiento es Random Forest utilizando SMOTE + ENN, sin

embargo, se observa que el algoritmo de vecinos próximos tras aplicar ambos algoritmos de *oversampling* y *undersampling* conjuntos, mejora en rendimiento medido con MRTN hasta valores parecidos al mejor rendimiento. En la sección 4.5 se puede observar la fusión de estos resultados con el rendimiento de los clasificadores con los resultados finales.
En la base de datos UNSW-NB15 se observa que el rendimiento, si bien mejora ligeramente, alcanza su mejor punto al aplicar *Random Forest* junto con SMOTE + ENN.

4.3 Enfoque de clase única (One Class Classification)

Una vez finalizadas las fases comunes definidas en el apartado 0 y obtenidos los conjuntos de datos finales con los que trabajar, se procede a realizar las fases correspondientes al enfoque de clasificación de clase única que solo tiene la fase de estudio de las métricas con los diferentes clasificadores de clase única.
Tras entrenar cada uno de los algoritmos y validarlos, se obtienen los siguientes resultados para cada base de datos.

	CSIC HTTP 2010			UNSW-NB15		
	Recall	TNR	MRTN	Recall	TNR	MRTN
Elliptic Envelope	0.6257	0.7432	0.6794	0.2249	0.8735	0.3577
Isolation Forest	0.5389	0.7457	0.6253	0.978	0.8735	**0.9228**
Local Outlier Factor	0.9423	0.8895	**0.9152**	0.6402	0.8639	0.7354
Autoencoder	0.6710	0.6896	0.6807	0.1783	0.6305	0.2779

Tabla 10. Métricas CSIC HTTP 2010 *OneClassClassification*.

Cómo podemos observar, para la base de datos CSIC HTTP 2010 el mejor clasificador de clase única en términos de MRTN es *Local Outlier Factor* con diferencia con respecto a los demás y para la base de datos UNSW-NB15 el mejor clasificador es *Isolation Forest* con gran diferencia frente a los demás también. De estos resultados se observa que el mejor método depende de la base de datos estudiada.
Cabe destacar que los métodos Elliptic Envelope y Autoencoder no tienen un rendimiento aceptable en ninguna de las dos bases de datos.

4.4 Comparación de resultados del estudio de complejidad de las bases de datos

Tras realizar el estudio de la complejidad de las bases de datos, se observa en la Tabla 4 (complejidad de CSIC HTTP 2010) y la Tabla 6 (complejidad de UNSW-NB15) que la complejidad de los datos de la base de datos UNSW-NB15 es claramente menor que la complejidad de CSIC HTTP 2010, ya que observando el discriminante de Fisher se aprecia que UNSW-NB15 tiene un atributo muy discriminante (sttl) el cual facilita la clasificación en gran medida.
Si se tienen en cuenta estos dos valores, se puede concluir que la base de datos CSIC HTTP 2010 tiene una complejidad alta y la base de datos UNSW-NB15 tienen una complejidad baja.

4.5 Resumen de resultados

Finalmente, los mejores resultados obtenidos para cada una de las bases de datos completas en cada uno de los enfoques son los siguientes. Es necesario tener en cuenta que la base de datos CSIC HTTP 2010 tiene un menor desbalanceo de clases (alrededor del 25% de anomalías) pero una complejidad muy alta y la base de datos UNSW-NB15 tiene un

desbalanceo de clases mayor (alrededor del 12% de anomalías) pero tiene una complejidad muy baja en comparación con la otra base de datos.

4.5.1 Resultados CSIC HTTP 2010 (complejidad alta)

Enfoque	Clasificador	Recall	TNR	MRTN	Accuracy	Precision	F1Score
Normal	*DecisionTree*	0.8607	0.965	0.9099	0.9331	0.8931	0.8701
Oversampling	***RandomForest + BorderlineSMOTE***	**0.8713**	**0.9647**	**0.9156**	**0.9405**	**0.8956**	**0.8832**
Oversampling + undersampling	*KNeighbors + SMOTE + Tomek Links*	0.9098	0.9119	0.9109	0.9114	0.7826	0.8414
Undersampling	*DecisionTree + Random Undersampler*	0.8877	0.9411	0.9136	0.9273	0.8400	0.8632
One Class Classification	***LocalOutlierFactor***	**0.9423**	**0.8895**	**0.9152**	**0.9032**	**0.7481**	**0.8340**

Tabla 11. Resultados de clasificadores con mejor rendimiento (en MRTN) por enfoques en CSIC HTTP 2010.

En esta tabla podemos observar que el enfoque que maximiza la métrica MRTN es el enfoque de *oversampling* con *RandomForest + BorderlineSMOTE* obteniendo un recall de 0.8713 y un TNR de 0.9647, muy de la mano del enfoque de clasificación de clase única dónde obtenemos un recall de 0.9423 y un TNR de 0.8895 mientras que el peor enfoque es el de clasificar sin aplicar ninguna medida extra para el desbalanceo de clases aunque sus métricas siguen estando en la línea de las mejores.

4.5.2 Resultados UNSW-NB15 (complejidad baja)

Enfoque	Clasificador	Recall	TNR	MRTN	Accuracy	Precision	F1Score
Normal	*RandomForest*	0.9735	0.9961	0.9847	0.9932	0.9734	0.9735
Oversampling	*RandomForest + ASADYN*	0.9859	0.9932	0.9895	0.9923	0.9547	0.9700
Oversampling + undersampling	***RandomForest + SMOTE + ENN***	**0.996**	**0.9887**	**0.9923**	**0.9896**	**0.9273**	**0.9604**
Undersampling	*RandomForest + Random Undersampler*	0.9936	0.9904	0.9920	0.9908	0.9376	0.9648
One Class Classification	*Isolation Forest*	0.978	0.8735	0.9228	0.8867	0.5283	0.6860

Tabla 12. Resultados de clasificadores con mejor rendimiento (en MRTN) por enfoques en UNSW-NB15.

En esta tabla podemos observar que el enfoque que maximiza la métrica MRTN es el enfoque de *oversampling + undersampling* con *RandomForest + SMOTE + ENN* obteniendo un recall de 0.996 y un TNR de 0.9887, cercanos a los valores de clasificación de del enfoque *undersampling* dónde obtenemos un recall de 0.9936 y un TNR de 0.9904.
El peor caso es el del enfoque de clasificación multiclase dónde el recall se mantiene alto (0.978), pero el TRN se reduce considerablemente en comparación con el resto de los valores hasta el valor de 0.8735.

4.5.3 Resultados de la métrica MRTN con umbrales de Scikit-learn

Tras obtener todos los resultados comentados previamente, se ha realizado un pequeño estudio comprobando la idoneidad de los umbrales de los algoritmos de Scikit-learn para la detección de anomalías. Para esto, se han probado, a modo ilustrativo, los diferentes algoritmos de clasificación multiclase utilizados en la base de datos CSIC HTTP 2010 sin aplicar ningún método de *oversampling* o *undersampling*, observando el punto de la curva ROC que elige el algoritmo de Scikit-learn y el punto en el cual la métrica MRTN es máxima. Los resultados se pueden observar en las siguientes figuras: Figura 10 y figuras en Anexo Figura 11 y Figura 12.

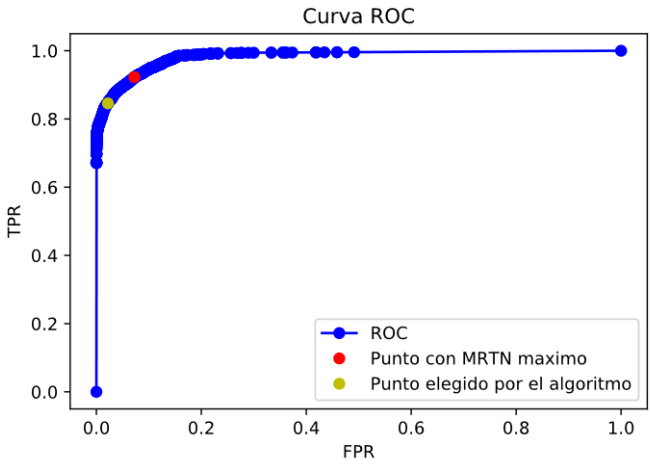

Figura 10. Curva ROC Random Forest.

Cómo se puede observar, los puntos elegidos por los algoritmos de clasificación de Scikit-learn no son los puntos óptimos para la detección de anomalías ya que, como se ve, los puntos que eligen por defecto los umbrales definidos en los algoritmos priorizan tener una proporción de falsos positivos baja a costa de perder proporción de falsos negativos.
Por esto, utilizando la medida MRTN se puede realizar un estudio en la curva ROC de los clasificadores con el fin de encontrar un umbral óptimo para la detección de anomalías en cada clasificador.
Este experimento se ha realizado al finalizar el trabajo y con el propósito de lanzar una línea de trabajo fututo, por lo que el ajuste de umbrales queda fuera del alcance del trabajo. Utilizando esta línea de trabajo futuro se podrían afinar aún más los resultados de la validación de este trabajo.

5 Conclusiones, discusión y trabajo futuro

5.1 Conclusiones y discusión

Finalmente, se puede concluir, a vista previa, que todos los objetivos de este trabajo se han cumplido con satisfacción.

También se puede concluir, que dentro del enfoque de aplicación de técnicas de *undersampling* y *oversampling*, los valores obtenidos de salida de la validación de cada uno de los sub enfoques (solo *oversampling*, solo *undersampling* o una mezcla de ambos) mejoran los resultados de la detección de anomalías tal y como se ha mostrado en la sección 4 y con detalles numéricos en las tablas en los Anexos A.D, A.E y A.F. Principalmente se observa que los algoritmos de *oversampling* son los que mayor mejora en la detección de anomalías aportan y cuanto mayor es en nivel de desbalanceo entre anomalías y elementos normales, mayor es la mejora que aportan estos algoritmos. También se observa que todos los algoritmos aplicados mejoran ligeramente los valores obtenidos sin aplicar dichas técnicas tanto con la base de datos completa, como con los conjuntos de datos generados eliminando elementos anómalos, a excepción del algoritmo de *undersampling* NearMiss que empeora considerablemente los resultados de la validación tras aplicarlo sobre las bases de datos.

Cabe destacar como conclusión la mejora que aportan estos algoritmos especialmente al aplicar el algoritmo de clasificación de vecinos próximos donde se encuentran las mejoras más considerables en la clasificación como se puede observar concretamente en las figuras y tablas comentadas.

También se puede concluir, observando los resultados finales de la base de datos UNSW-NB15 en la sección 4 y con detalles numéricos en las tablas de los Anexos A.D.2 y A.E.2, que, si bien todo el enfoque de oversampling y undersampling mejora el rendimiento, aplicar la clasificación de clase única reduce este rendimiento por lo cual se propone como trabajo futuro un estudio más profundo desde este enfoque para abordar esta base de datos, principalmente abarcando diferentes configuraciones de autoencoder.

Observando los resultados finales, y observando la complejidad de los datos de cada una de las bases de datos en las tablas Tabla 4 y Tabla 6, también se concluye que cuanto mayor es la complejidad de los datos, mayor es la dificultad del problema a la hora de clasificar anomalías, ya que la base de datos HTTP CSIC 2010 tiene una complejidad alta en comparación con la base de datos UNSW-NB15, y los resultados de la validación de los clasificadores son considerablemente mejores en la base de datos UNSW-NB15 que en la base de datos CSIC HTTP 2010.

Junto con la conclusión anterior, podemos afirmar también que en la base de datos UNS-NB15 completa (Tabla 12) haciendo uso de ***RandomForest + BorderlineSMOTE*** se han detectado casi en su totalidad las anomalías presentes y casi en su totalidad todos los elementos normales. Sin embargo, en la base de datos CSIC HTTP 2010 completa (Tabla 11) haciendo uso de ***RandomForest + SMOTE + ENN*** o bien se decide pasar por alto alrededor de un 11% de anomalías, con un TNR del 94% o bien, haciendo uso de ***LocalOutlierFactor***, se decide dejar pasar por alto un 10.5% de los elementos normales como falsas alarmas y obteniendo solo un 3.5% de falsos negativos (utilizando ambas configuraciones el valor de MRTN obtenido es muy similar).

Siguiendo con el hilo anterior, y de cara a tomar decisiones sobre qué enfoque elegir una vez obtenidas las métricas y disponiendo de diferencias tan pequeñas como las presentes, se necesitaría disponer del coste de tomar cada una de las decisiones (coste de una falsa alarma

y coste de una anomalía no detectada) con el fin de poder decidir correctamente qué elemento considerar primordialmente.
Cabe destacar que tanto a nivel de aplicación (tráfico HTTP) como a nivel de red (flujos de red) *Random Forest* y Árboles de decisión son buenos métodos que aplicar para detección de anomalías sin aplicar ninguna corrección para paliar el efecto del desbalanceo de clases ya que ambos algoritmos son los que mejor rendimiento, en términos de MRTN, obtienen.
Como era de esperar, en la base de datos UNSW-NB15 se obtienen unos resultados mejores ya que la complejidad de los datos de esta base de datos es baja y en la base de datos CSIC HTTP 2010 se obtienen resultados buenos, pero peores en comparación a los de la otra base de datos ya que la complejidad de los datos es alta.
La complejidad de los datos, si bien para una única base de datos no es ilustrativa ya que carece de comparación, una vez se ha obtenido la complejidad de una base de datos y se ha observado que los resultados en la detección de anomalías son buenos, se pueden establecer las medidas de complejidad de esa base de datos como elemento de referencia para el estudio posterior de la complejidad de otras bases de datos. Por esto, una vez obtenida la complejidad de un problema resuelto como elemento de comparación, se puede estimar el grado de confianza para resolver un nuevo problema estudiando la complejidad de los datos del nuevo problema.

5.2 Trabajo futuro

Como trabajo futuro, se propone la investigación de nuevas bases de datos con ambos enfoques para aumentar la evidencia empírica, y también se propone la posibilidad de crear una nueva base de datos con datos no generados artificialmente para el mismo fin.
Otra línea de estudio para trabajo futuro propuesta es el estudio de la influencia en la complejidad de los datos al aplicar métodos de *oversampling* y *undersampling*.
También se propone como posible trabajo futuro el estudio de diferentes combinaciones de *autoencoders* para comprobar si existen otros que se adecuen mejor a los problemas presentados.
Otra propuesta de trabajo futuro posible es la del estudio de los umbrales mediante el uso de la métrica MRTN con el fin de obtener el umbral óptimo según esta medida para la detección de anomalías, así como el estudio empírico de su eficacia.
Por último, se propone el estudio de como combinar datos de las capas 4 y 7 del modelo OSI con el fin de mejorar la detección de anomalías.

Referencias

[1] Julian Jang-Jaccard, Surya Nepal, “A survey of emerging threats in cybersecurity”, Journal of Computer and System Sciences, Volume 80, Issue 5, 2014, Pages 973-993.

[2] Mohan V. Pawar, J. Anuradha, “Network Security and Types of Attacks in Network”, Procedia Computer Science, Volume 48, 2015, Pages 503-506.

[3] Daljit Kaur, Parminder Kaur, "Empirical Analysis of Web Attacks", Procedia Computer Science, Volume 78, 2016, Pages 298-306.

[4] Valdes A., Skinner K. “Adaptive, Model-Based Monitoring for Cyber Attack Detection”. Debar H., Mé L., Wu S.F. (eds) Recent Advances in Intrusion Detection. RAID 2000. Lecture Notes in Computer Science, vol 1907, 2000 Springer, Berlin, Heidelberg

[5] Alhajri, Reem, Rachid Zagrouba, and Fahd Al-Haidari. "Survey for Anomaly Detection of IoT Botnets Using Machine Learning Auto-Encoders." International Journal of Applied Engineering Research 14.10 (2019): 2417-2421.

[6] Nguyen, Giang & Dlugolinsky, Stefan & Tran, Viet & Lopez Garcia, Alvaro. (2020). Deep learning for proactive network monitoring and security protection. IEEE Access. PP. 1-1.

[7] Darktrace, Tecnología. 01/06/2020 https://www.darktrace.com/es/tecnolog%C3%ADa/

[8] Ring, Markus, et al. “A survey of network-based intrusion detection data sets”, Computers & Security, 2019.

[9] Kotsiantis, Sotiris & Kanellopoulos, D. & Pintelas, P.. (2005). Handling imbalanced datasets: A review. GESTS International Transactions on Computer Science and Engineering. 30. 25-36.

[10] Bora, Gaurav, et al. "OSI reference model: An overview." International Journal of Computer Trends and Technology (IJCTT) 7.4 (2014): 214-218.

[11] C. Torrano-Gimenez, Study of stochastic and machine learning techniques for anomaly-based web attack detection, 2015.

[12] Moustafa, Nour, and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." Military Communications and Information Systems Conference (MilCIS), 2015. IEEE, 2015.

[13] JM Sotoca, JS Sánchez, RA Mollineda. A review of data complexity measures and their applicability to pattern classification problems. Actas del III Taller Nacional de Mineria de Datos y Aprendizaje. TAMIDA, 77-83

[14] Tin Kam Ho and M. Basu. "Complexity measures of supervised classification problems,". IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 3, pp. 289-300, March 2002, doi: 10.1109/34.990132.

[15] OWASP Foundation, SQL Injection. 09/05/2020 https://owasp.org/www-community/attacks/SQL_Injection

[16] OWASP Foundation, *Buffer Overflow*. 09/05/2020 https://owasp.org/www-community/attacks/Buffer_overflow_attack

[17] W3Schools, *Information Gathering Techniques*, 09/05/2020 https://www.w3schools.in/ethical-hacking/information-gathering-techniques/

[18] Ahmed, M. & Hassan, Md Maruf & Bhuyian, Touhid. (2018). Local File Disclosure Vulnerability: A Case Study of Public-Sector Web Applications. Journal of Physics: Conference Series. 933. 012011. 10.1088/1742-6596/933/1/012011.

[19] OWASP Foundation, CRLF Injection, 09/05/2020 https://owasp.org/www-community/vulnerabilities/CRLF_Injection

[20] Isatou Hydara, Abu Bakar Md. Sultan, Hazura Zulzalil, Novia Admodisastro, Current state of research on cross-site scripting (XSS) – A systematic literatura review, Information and Software Technology, Volume 58, 2015, Pages 170-186, ISSN 0950-5849, https://doi.org/10.1016/j.infsof.2014.07.010.
[21] OWASP Foundation, OWASP Top Ten, 09/05/2020 https://owasp.org/www-project-top-ten/
[22] OWASP Foundation, Server Side Includes, 09/05/2020 https://owasp.org/www-community/attacks/Server-Side_Includes_(SSI)_Injection
[23] OWASP Foundation, Web Parameter Tampering, 09/05/2020 https://owasp.org/www-community/attacks/Web_Parameter_Tampering
[24] INCIBE, Amenaza vs Vulnerabilidad, 09/05/2020 https://www.incibe.es/protege-tu-empresa/blog/amenaza-vs-vulnerabilidad-sabes-se-diferencian
[25] Patrick Engebretson, Chapter 2 - Reconnaissance, Editor(s): Patrick Engebretson, The Basics of Hacking and Penetration Testing, Syngress, 2011, Pages 15-41, ISBN 9781597496551, https://doi.org/10.1016/B978-1-59749-655-1.00002-7.
[26] Palo Alto Networks, What is a Denial Of Service DoS, 09/05/2020 https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos
[27] N. V. Chawla, K. W. Bowyer, L. O.Hall, W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," Journal of artificial intelligence research, 321-357, 2002.
[28] Santos, Miriam & Soares, Jastin & Henriques Abreu, Pedro & Araujo, Helder & Santos, Joao. (2018). Cross-Validation for Imbalanced Datasets: Avoiding Overoptimistic and Overfitting Approaches. IEEE Computational Intelligence Magazine. 13. 59-76. 10.1109/MCI.2018.2866730.
[29] D. L. Wilson, "Asymptotic properties of nearest neighbour rules using edited data," IEEE Transactions on Systems, Man, and Cybernetics, vol. 2, no. 3, pp. 408–421, July 1972
[30] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li, ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning.
[31] Han H., Wang WY., Mao BH. (2005) Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In: Huang DS., Zhang XP., Huang GB. (eds) Advances in Intelligent Computing. ICIC 2005. Lecture Notes in Computer Science, vol 3644. Springer, Berlin, Heidelberg
[32] Zhang, J.P. and Mani, I. (2003) KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction. Proceeding of International Conference on Machine Learning (ICML 2003)
[33] G. Batista, R. C. Prati, M. C. Monard. "A study of the behavior of several methods for balancing machine learning training data," ACM Sigkdd Explorations Newsletter 6 (1), 20-29, 2004.
[34] G. Batista, B. Bazzan, M. Monard, "Balancing Training Data for Automated Annotation of Keywords: a Case Study," In WOB, 10-18, 2003
[35] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and Regression Trees", Wadsworth, Belmont
[36] Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.
[37] Jacob Goldberger, Sam Roweis, Geoff Hinton, Ruslan Salakhutdinov. "Neighbourhood Components Analysis"
[38] Liu, Fei Tony, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation forest." 2008 Eighth IEEE International Conference on Data Mining. IEEE, 2008.
[39] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2012. Isolation-Based Anomaly Detection. ACM Trans. Knowl. Discov. Data 6, 1, Article 3 (March 2012).

[40] Scikit-learn, Outlier Detection. 09/05/2020 https://scikit-learn.org/stable/modules/outlier_detection.html
[41] Rousseeuw, Peter & Driessen, Katrien. (1999). A Fast Algorithm for the Minimum Covariance Determinant Estimator. Technometrics. 41. 212-223.
[42] Breunig, M. M., Kriegel, H., Ng, R. T., & Sander, J. (2000). LOF: Identifying Density-Based Local Outliers. pp 1-12.
[43] W. Wang, Y. Huang, Y. Wang and L. Wang, "Generalized Autoencoder: A Neural Network Framework for Dimensionality Reduction," 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops, Columbus, OH, 2014, pp. 496-503
[44] Wenjun Sun, Siyu Shao, Rui Zhao, Ruqiang Yan, Xingwu Zhang, Xuefeng Chen, A sparse auto-encoder-based deep neural network approach for induction motor faults classification, Measurement, Volume 89, 2016, Pages 171-178
[45] Scikit-learn, Supervised Classification. 10/05/2020 https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
[46] Bro-IDS (Zeek), 13/05/2020 https://zeek.org
[47] Argus, openargusm 13/05/2020 https://openargus.org/
[48] Tcpdump, Tcpdump Manpage, 13/05/2020 https://www.tcpdump.org/manpages/tcpdump.1.html
[49] Ixia, Ixia PerfectStorm tool, 14/05/2020 http://www.ixiacom.com/products/perfectstorm

Glosario

FPR	*False Positive Rate.*
TNR	*True Negative Rate.*
MRTN	*Maximal Recall and TNR.*
SMOTE	*Synthetic Minority Over-sampling TEchnique*
ADASYN	*Adaptative Synthetic*

Anexos

A Ejemplos de entradas en las bases de datos

Se muestra un ejemplo en formato plano de cada una de las bases de datos antes de ser procesada y después de ser procesada (y tal y como se utilizará en los experimentos)

A.1 Base de datos CSIC HTTP 2010

(1)Antes del procesamiento

Ejemplo de petición normal:

```
GET http://localhost:8080/tienda1/index.jsp HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux)
KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/p
lain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=1F767F17239C9B670A39E9B10C3825F4
Connection: close
```

Ejemplo de petición anómala:

```
POST http://localhost:8080/tienda1/publico/anadir.jsp HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux)
KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/p
lain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=AE29AEEBDE479D5E1A18B4108C8E3CE0
Content-Type: application/x-www-form-urlencoded
Connection: close
Content-Length: 146

id=2&nombre=Jam%F3n+Ib%E9rico&precio=85&cantidad=%27%3B+DROP+TABLE+us
uarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25&B1=A%F1adir+al
+carrito
```

(2)Después del procesamiento

Ejemplo de petición normal:

- req_lengths: 478
- path_lenghts: 38
- arg_lenghts: 60
- arg_letters_num: 50
- arg_digits_num: 1
- arg_special_num: 9
- path_letter_num: 30
- distinct_bytes: 64
- d: 2
- 3: 0
- p: 2
- -: 0
- i: 4
- t: 2
- P: 0
- label: 0

Ejemplo de petición anómala:

- req_lengths: 557
- path_lenghts: 34
- arg_lenghts: 146
- arg_letters_num: 99
- arg_digits_num: 15
- arg_special_num: 32
- path_letter_num: 26
- distinct_bytes: 68
- d: 5
- 3: 3
- p: 1
- -: 0
- i: 7
- t: 3
- P: 1
- label: 1

A.2 Base de datos UNSW-NB15

(1)Antes del procesamiento

- srcip: 59.166.0.0
- sport: 1390
- dstip: 149.171.126.6
- dsport: 53
- proto: udp
- state: CON
- dur: 0.001055

- sbytes: 132
- dbytes: 164
- sttl: 31
- dttl: 29.0
- sloss: 0.0
- dloss: 0.0
- service: dns
- Sload: 500473.93750
- Dload: 621800.93750
- Spkts: 2.0
- Dpkts: 2.0
- swin: 0.0
- dwin: 0.0
- stcpb: 0.0
- dtcpb: 0.0
- smeansz: 66.0
- dmeansz: 82.0
- trans_depth: 0.0
- res_bdy_len: 0.0
- Sjit: 0.00000
- Djit: 0.000000
- Stime: 1.421927e+09
- Ltime: 1.421927e+09
- Sintpkt: 0.017
- Dintpkt: 0.013000
- tcprtt: 0.0
- synack: 0.0
- ackdat: 0.0
- is_sm_ips_ports: 0.0
- ct_state_ttl: 0.0
- ct_flw_http_mthd: 0.0
- is_ftp_login: 0.0
- ct_ftp_cmd: 0
- ct_srv_src: 3.0
- ct_srv_dst: 7.0
- ct_dst_ltm: 1.0
- ct_src_ ltm: 3.0
- ct_src_dport_ltm: 1.0
- ct_dst_sport_ltm: 1.0
- ct_dst_src_ltm: 1.0
- attack_cat: NaN
- Label: 0

(2)Después del procesamiento

- sbytes: 132
- sttl: 31
- Sload: 500473.93750

- Spkts: 2.0
- smeansz: 66.0
- synack: 0.0
- ct_srv_dst: 7.0
- Label: 0

B Peticiones HTTP

Las peticiones HTTP son las solicitudes o mensajes que envían los clientes a servidores HTTP cuando este le solicita un recurso.
Estas peticiones contienen los siguientes elementos:

- Método: Indica el tipo de petición que se realiza (puede ser de varios tipos, los más comunes son GET y POST).
- Ruta: Indica la ruta del recurso solicitado al servidor.
- Protocolo: Indica el protocolo mediante el cual se realiza la petición junto con la versión del protocolo.
- Cabeceras: Se trata de una lista de pares clave-valor con diferentes elementos que pueden ser necesarios por el servidor para tratar correctamente la petición. Por ejemplo, el par User-Agent: Valor indica comúnmente el navegador desde el cual se está realizando dicha petición.
- Cuerpo de la petición: En caso de que el cliente necesite enviar información adicional al servidor (por ejemplo, datos de un formulario relleno), estos valores se indicarán en el cuerpo del mensaje en caso de que el método indicado sea POST. Si el método indicado es GET, esta información en forma de parámetros irá concatenada a la ruta de la petición.

En el ejemplo de petición anómala del Anexo A.1(1), estos elementos se diferencian así:

Método: `POST`
Ruta: `http://localhost:8080/tienda1/publico/anadir.jsp`
Protocolo: `HTTP/1.1`

Cabeceras:

```
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux)
KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/p
lain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=AE29AEEBDE479D5E1A18B4108C8E3CE0
Content-Type: application/x-www-form-urlencoded
Connection: close
Content-Length: 146
```

Cuerpo de la petición:

```
id=2&nombre=Jam%F3n+Ib%E9rico&precio=85&cantidad=%27%3B+DROP+TABLE+us
uarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25&B1=A%F1adir+al
+carrito
```

C Flujos de tráfico de red

Los flujos de red son agrupaciones de paquetes pertenecientes a una misma comunicación entre cliente y el servidor en un intervalo determinado de tiempo. Estos flujos se obtienen de procesar los ficheros en formato pcap que contienen todo el tráfico analizado por la herramienta *tcpdump* en las interfaces de cada uno de los servidores que reciben el tráfico.

D Comparativas con algoritmos de oversampling

Se muestran a continuación los datos relacionados con la aplicación de algoritmos de *oversampling* sobre cada una de las bases de datos.

En todas las tablas comparativas a continuación se muestran los resultados para cada desbalanceo estudiado de la validación utilizando cada clasificador y cada algoritmo de oversampling o undersampling. En cada fila se muestran los valores de recall, TNR y MRTN tanto aplicando el algoritmo, como sin aplicarlo (columnas "Normal").

D.1 CSIC HTTP 2010

Classifier	Oversampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE	0.6888	0.659	0.9979	0.9968	0.8141	0.7931
DecisionTree	BorderlineSMOTE	0.7153	0.6745	0.9976	0.9975	0.833	0.8043
DecisionTree	ADASYN	0.7194	0.6517	0.9977	0.997	**0.836**	0.7874
RandomForest	SMOTE	0.6518	0.6824	0.9996	0.9989	0.7885	0.8104
RandomForest	BorderlineSMOTE	0.6823	0.6988	0.9997	0.9994	0.811	0.8223
RandomForest	ADASYN	0.6389	0.6326	0.9997	0.9993	0.7791	0.7742
KNeighbors	SMOTE	0.5102	0.7878	0.9999	0.9649	0.6748	**0.8672**
KNeighbors	BorderlineSMOTE	0.5289	0.7252	1.0	0.9823	0.6912	0.8341
KNeighbors	ADASYN	0.5356	0.7871	1.0	0.9582	0.6968	0.864

Tabla 13. Métricas CSIC HTTP 2010 con oversampling en 1%.

Classifier	Oversampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE	0.7569	0.7472	0.9938	0.9924	0.8591	0.8523
DecisionTree	BorderlineSMOTE	0.7606	0.7499	0.9944	0.9917	**0.8619**	0.8539
DecisionTree	ADASYN	0.7535	0.7301	0.9937	0.9923	0.857	0.841
RandomForest	SMOTE	0.7315	0.7461	0.9988	0.9973	0.8445	0.8535
RandomForest	BorderlineSMOTE	0.7213	0.7297	0.9988	0.9967	0.8376	0.8425
RandomForest	ADASYN	0.7427	0.7463	0.9988	0.9971	0.8518	0.8535
KNeighbors	SMOTE	0.6509	0.8254	0.9998	0.9513	0.7883	0.8838
KNeighbors	BorderlineSMOTE	0.6654	0.8154	0.9968	0.9592	0.798	0.8814
KNeighbors	ADASYN	0.6531	0.8549	0.9998	0.9378	0.7899	**0.8944**

Tabla 14. Métricas CSIC HTTP 2010 con oversampling en 3.22%.

Classifier	Oversampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE	0.7902	0.7834	0.9887	0.9871	0.8784	0.8735
DecisionTree	BorderlineSMOTE	0.7927	0.7747	0.988	0.9856	0.8796	0.8675
DecisionTree	ADASYN	0.7984	0.7856	0.9889	0.9865	**0.8835**	0.8747
RandomForest	SMOTE	0.7648	0.7779	0.9974	0.9953	0.8658	0.8732
RandomForest	BorderlineSMOTE	0.765	0.7764	0.9973	0.995	0.8658	0.8722

RandomForest	ADASYN	0.7534	0.7624	0.9976	0.995	0.8584	0.8633
KNeighbors	SMOTE	0.7059	0.8609	0.9983	0.9411	0.827	0.8992
KNeighbors	BorderlineSMOTE	0.7042	0.8668	0.9925	0.932	0.8237	0.8982
KNeighbors	ADASYN	0.7071	0.8841	0.9965	0.9235	0.8272	**0.9034**

Tabla 15. Métricas CSIC HTTP 2010 con oversampling en 6.45%.

Classifier	Oversampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE	0.8195	0.8181	0.9798	0.9779	0.8925	0.8909
DecisionTree	BorderlineSMOTE	0.8224	0.8183	0.9791	0.9749	0.8938	0.8897
DecisionTree	ADASYN	0.8305	0.8241	0.9797	0.9725	**0.8989**	0.8922
RandomForest	SMOTE	0.7933	0.8099	0.9931	0.9881	0.882	0.8901
RandomForest	BorderlineSMOTE	0.7892	0.8078	0.9936	0.9864	0.8797	0.8882
RandomForest	ADASYN	0.7876	0.8083	0.993	0.9839	0.8784	0.8875
KNeighbors	SMOTE	0.7426	0.8872	0.9885	0.9296	0.8481	**0.9079**
KNeighbors	BorderlineSMOTE	0.7439	0.9075	0.9893	0.8993	0.8492	0.9034
KNeighbors	ADASYN	0.7472	0.9213	0.9899	0.8978	0.8516	0.9093

Tabla 16. Métricas CSIC HTTP 2010 con oversampling en 12.91%.

Classifier	Oversampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE	0.86	0.8593	0.9646	0.9596	0.9093	0.9067
DecisionTree	BorderlineSMOTE	0.8607	0.8671	0.965	0.9561	**0.9099**	0.9094
DecisionTree	ADASYN	0.8611	0.8654	0.9646	0.9535	**0.9099**	0.9073
RandomForest	SMOTE	0.8459	0.861	0.9789	0.9719	0.9076	0.9131
RandomForest	BorderlineSMOTE	0.8458	0.8713	0.9788	0.9647	0.9075	**0.9156**
RandomForest	ADASYN	0.8452	0.872	0.9789	0.963	0.9071	0.9152
KNeighbors	SMOTE	0.8143	0.9087	0.9593	0.9163	0.8808	0.9125
KNeighbors	BorderlineSMOTE	0.8143	0.9464	0.9593	0.8694	0.8808	0.9062
KNeighbors	ADASYN	0.8143	0.9501	0.9593	0.8627	0.8808	0.9043

Tabla 17. Métricas CSIC HTTP 2010 con oversampling en 25.82%.

D.2 UNSW-NB15

	Oversampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE	0.9065	0.915	0.9991	0.9986	0.9506	0.955
DecisionTree	BorderlineSMOTE	0.9077	0.915	0.9992	0.9987	0.9512	0.955
DecisionTree	ADASYN	0.9069	0.9155	0.9992	0.9986	0.9508	0.9552
RandomForest	SMOTE	0.9055	0.9331	0.9998	0.9986	0.9503	0.9647
RandomForest	BorderlineSMOTE	0.9078	0.9296	0.9998	0.9989	0.9516	0.963
RandomForest	ADASYN	0.9087	0.9364	0.9998	0.9986	**0.9521**	**0.9665**
KNeighbors	SMOTE	0.7965	0.9246	0.9997	0.9914	0.8866	0.9569
KNeighbors	BorderlineSMOTE	0.7952	0.9004	0.9997	0.994	0.8858	0.9449
KNeighbors	ADASYN	0.7959	0.9329	0.9997	0.9899	0.8862	0.9605

Tabla 18. Métricas UNSW-NB15 con oversampling en 1%.

	Oversampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE	0.9212	0.9316	0.9989	0.9982	**0.9585**	0.9638

DecisionTree	BorderlineSMOTE	0.921	0.928	0.9989	0.9982	0.9584	0.9618
DecisionTree	ADASYN	0.9211	0.9305	0.9989	0.9982	0.9584	0.9632
RandomForest	SMOTE	0.918	0.9426	0.9996	0.9982	0.9571	0.9696
RandomForest	BorderlineSMOTE	0.9169	0.9397	0.9996	0.9983	0.9565	0.9681
RandomForest	ADASYN	0.9189	0.9432	0.9996	0.9981	0.9575	**0.9699**
KNeighbors	SMOTE	0.8167	0.9383	0.9995	0.9907	0.8989	0.9638
KNeighbors	BorderlineSMOTE	0.8125	0.9235	0.9995	0.992	0.8963	0.9565
KNeighbors	ADASYN	0.8157	0.9472	0.9995	0.9887	0.8983	0.9675

Tabla 19. Métricas UNSW-NB15 con oversampling en 1.58%.

	Oversampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE	0.9348	0.9456	0.9982	0.9972	0.9655	0.9707
DecisionTree	BorderlineSMOTE	0.9357	0.9436	0.9982	0.9972	0.9659	0.9697
DecisionTree	ADASYN	0.9358	0.9445	0.9982	0.9971	0.966	0.9701
RandomForest	SMOTE	0.9368	0.9583	0.9992	0.9973	0.967	0.9774
RandomForest	BorderlineSMOTE	0.9341	0.9579	0.9992	0.9971	0.9655	0.9771
RandomForest	ADASYN	0.9351	0.959	0.9992	0.9971	**0.9661**	**0.9777**
KNeighbors	SMOTE	0.8541	0.9549	0.9987	0.9894	0.9208	0.9718
KNeighbors	BorderlineSMOTE	0.8551	0.9517	0.9986	0.9895	0.9213	0.9702
KNeighbors	ADASYN	0.8557	0.9633	0.9987	0.9865	0.9217	0.9748

Tabla 20. Métricas UNSW-NB15 con oversampling en 3.16%.

	Oversampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE	0.9517	0.9602	0.9971	0.996	0.9739	0.9778
DecisionTree	BorderlineSMOTE	0.9517	0.9599	0.9971	0.996	0.9739	0.9776
DecisionTree	ADASYN	0.9516	0.9601	0.9972	0.9959	0.9739	0.9777
RandomForest	SMOTE	0.9536	0.9715	0.9982	0.996	0.9754	0.9836
RandomForest	BorderlineSMOTE	0.9533	0.9715	0.9982	0.9955	0.9752	0.9834
RandomForest	ADASYN	0.9539	0.9736	0.9982	0.9954	**0.9756**	**0.9844**
KNeighbors	SMOTE	0.8948	0.9682	0.9968	0.9877	0.9431	0.9779
KNeighbors	BorderlineSMOTE	0.8947	0.971	0.9968	0.9863	0.943	0.9786
KNeighbors	ADASYN	0.8937	0.9777	0.9969	0.9842	0.9425	0.981

Tabla 21. Métricas UNSW-NB15 con oversampling en 6.32%.

	Oversampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE	0.968	0.974	0.9955	0.9945	0.9815	0.9841
DecisionTree	BorderlineSMOTE	0.9682	0.9726	0.9955	0.9943	0.9816	0.9833
DecisionTree	ADASYN	0.9681	0.9739	0.9954	0.9942	0.9816	0.9839
RandomForest	SMOTE	0.9736	0.983	0.9962	0.9943	0.9848	0.9886
RandomForest	BorderlineSMOTE	0.9735	0.9844	0.9962	0.9935	**0.9847**	0.9889
RandomForest	ADASYN	0.9736	0.9859	0.9962	0.9932	**0.9847**	**0.9896**
KNeighbors	SMOTE	0.9334	0.9783	0.9935	0.986	0.9625	0.9821
KNeighbors	BorderlineSMOTE	0.9334	0.9836	0.9935	0.9834	0.9625	0.9835
KNeighbors	ADASYN	0.9334	0.988	0.9935	0.9815	0.9625	0.9848

Tabla 22. Métricas UNSW-NB15 con oversampling en 12.64%.

E Comparativas con algoritmos de undersampling

Se muestran a continuación los datos relacionados con la aplicación de algoritmos de und*ersampling* sobre cada una de las bases de datos.

E.1 CSIC HTTP 2010

Classifier	Undersampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	RandomUnderSampler	0.6743	0.801	0.9978	0.9244	0.8044	**0.8578**
DecisionTree	NearMiss	0.6928	0.9932	0.9978	0.0581	0.8173	0.1086
RandomForest	RandomUnderSampler	0.6824	0.7925	0.9997	0.9448	**0.811**	0.8607
RandomForest	NearMiss	0.6627	0.9973	0.9997	0.0517	0.7968	0.0973
KNeighbors	RandomUnderSampler	0.4815	0.6192	1.0	0.9525	0.6499	0.7496
KNeighbors	NearMiss	0.5133	0.9821	1.0	0.0856	0.6782	0.156

Tabla 23. Métricas CSIC HTTP 2010 con undersampling en 1454 puntos.

Classifier	Undersampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	RandomUnderSampler	0.7522	0.8501	0.9939	0.9361	0.8563	**0.891**
DecisionTree	NearMiss	0.7628	0.9896	0.9937	0.1426	**0.8629**	0.2487
RandomForest	RandomUnderSampler	0.7426	0.8497	0.9988	0.9499	0.8519	0.897
RandomForest	NearMiss	0.7231	0.9958	0.9987	0.0893	0.8388	0.1639
KNeighbors	RandomUnderSampler	0.6559	0.7638	0.9998	0.955	0.7921	0.8488
KNeighbors	NearMiss	0.6413	0.9792	0.9998	0.1616	0.7814	0.2769

Tabla 24. Métricas CSIC HTTP 2010 con undersampling en 4802 puntos.

Classifier	Undersampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	RandomUnderSampler	0.7918	0.8696	0.9886	0.9399	**0.8793**	**0.9034**
DecisionTree	NearMiss	0.79	0.9882	0.9891	0.3201	0.8783	0.482
RandomForest	RandomUnderSampler	0.7596	0.8576	0.9972	0.9508	0.8623	0.9017
RandomForest	NearMiss	0.7559	0.9913	0.9973	0.1953	0.86	0.3253
KNeighbors	RandomUnderSampler	0.7002	0.789	0.9994	0.9576	0.8234	0.8651
KNeighbors	NearMiss	0.708	0.9747	0.9993	0.3015	0.8288	0.4602

Tabla 25.Métricas CSIC HTTP 2010 con undersampling en 9936 puntos.

Classifier	Undersampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	RandomUnderSampler	0.8227	0.8698	0.9789	0.9408	0.894	0.9039
DecisionTree	NearMiss	0.8218	0.9782	0.9807	0.495	**0.8943**	0.6525
RandomForest	RandomUnderSampler	0.7865	0.8668	0.9932	0.9538	0.8778	**0.9082**
RandomForest	NearMiss	0.7903	0.9877	0.9931	0.431	0.8802	0.595
KNeighbors	RandomUnderSampler	0.741	0.8191	0.9875	0.957	0.8467	0.8827
KNeighbors	NearMiss	0.7465	0.9627	0.992	0.5678	0.8519	0.714

Tabla 26. Métricas CSIC HTTP 2010 con undersampling en 21384 puntos.

Classifier	Undersampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	RandomUnderSampler	0.8603	0.8878	0.963	0.9412	0.9088	**0.9137**

DecisionTree	NearMiss	0.8612	0.9412	0.9628	0.7733	**0.9092**	0.847
RandomForest	RandomUnderSampler	0.8444	0.8895	0.9785	0.9506	0.9065	0.919
RandomForest	NearMiss	0.8427	0.9495	0.9784	0.7644	0.9055	0.8446
KNeighbors	RandomUnderSampler	0.8106	0.8453	0.9573	0.9561	0.8778	0.8973
KNeighbors	NearMiss	0.8106	0.9194	0.9573	0.774	0.8778	0.8384

Tabla 27. Métricas CSIC HTTP 2010 con undersampling en 50130 puntos.

E.2 UNSW-NB15

Classifier	Undersampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	RandomUnderSampler	0.9087	0.9811	0.9992	0.9907	0.9518	0.9858
DecisionTree	NearMiss	0.9114	0.9427	0.9992	0.3605	0.9532	0.5184
RandomForest	RandomUnderSampler	0.9099	0.9921	0.9998	0.9891	**0.9527**	**0.9906**
RandomForest	NearMiss	0.9042	0.942	0.9998	0.5849	0.9496	0.7069
KNeighbors	RandomUnderSampler	0.7928	0.9426	0.9997	0.9773	0.8843	0.9596
KNeighbors	NearMiss	0.7969	0.8914	0.9997	0.0677	0.8868	0.1258

Tabla 28. Métricas UNSW-NB15 con undersampling en 44822 puntos.

Classifier	Undersampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	RandomUnderSampler	0.9196	0.98	0.9988	0.9909	0.9576	0.9854
DecisionTree	NearMiss	0.9226	0.9441	0.9989	0.5277	**0.9593**	0.665
RandomForest	RandomUnderSampler	0.9193	0.9921	0.9996	0.9897	0.9578	**0.9909**
RandomForest	NearMiss	0.9178	0.9441	0.9997	0.9953	0.957	0.969
KNeighbors	RandomUnderSampler	0.8127	0.951	0.9995	0.9782	0.8965	0.9644
KNeighbors	NearMiss	0.8131	0.8898	0.9995	0.1294	0.8967	0.2257

Tabla 29. Métricas UNSW-NB15 con undersampling en 71288 puntos.

Classifier	Undersampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	RandomUnderSampler	0.9363	0.9829	0.9982	0.991	**0.9663**	0.9869
DecisionTree	NearMiss	0.9344	0.95	0.9982	0.7715	0.9652	0.8421
RandomForest	RandomUnderSampler	0.9354	0.9921	0.9992	0.9899	0.9662	**0.991**
RandomForest	NearMiss	0.9345	0.9528	0.9992	0.9344	0.9658	0.9419
KNeighbors	RandomUnderSampler	0.8546	0.9574	0.9987	0.9812	0.921	0.9691
KNeighbors	NearMiss	0.8561	0.9109	0.9987	0.1578	0.9219	0.2685

Tabla 30. Métricas UNSW-NB15 con undersampling en 144904 puntos.

Classifier	Undersampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	RandomUnderSampler	0.9514	0.9838	0.9972	0.9913	0.9737	0.9875
DecisionTree	NearMiss	0.9519	0.9616	0.9972	0.9949	0.974	0.978
RandomForest	RandomUnderSampler	0.9541	0.9928	0.9982	0.9901	**0.9757**	**0.9914**
RandomForest	NearMiss	0.9534	0.9635	0.9982	0.9949	0.9753	0.979
KNeighbors	RandomUnderSampler	0.8945	0.9642	0.9968	0.9831	0.9429	0.9736
KNeighbors	NearMiss	0.8938	0.9286	0.9969	0.1804	0.9425	0.302

Tabla 31. Métricas UNSW-NB15 con undersampling en 299592 puntos.

Classifier	Undersampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	RandomUnderSampler	0.9678	0.9863	0.9955	0.9913	0.9814	0.9888
DecisionTree	NearMiss	0.9681	0.9713	0.9955	0.9947	0.9816	0.9829
RandomForest	RandomUnderSampler	0.9735	0.9936	0.9962	0.9904	0.9847	**0.992**
RandomForest	NearMiss	0.9736	0.9763	0.9962	0.9954	**0.9848**	0.9858
KNeighbors	RandomUnderSampler	0.9334	0.9688	0.9935	0.9849	0.9625	0.9768
KNeighbors	NearMiss	0.9334	0.938	0.9935	0.396	0.9625	0.5569

Tabla 32. Métricas UNSW-NB15 con undersampling en 642566 puntos.

F Comparativas con algoritmos de undersampling y oversampling conjuntos

Se muestran a continuación los datos relacionados con la aplicación de algoritmos de *oversampling* y *undersampling* conjuntamente sobre cada una de las bases de datos.

F.1 CSIC HTTP 2010

Classifier	Oversampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE + ENN	0.6919	0.6782	0.9974	0.9896	**0.8168**	0.8047
DecisionTree	SMOTE + Tomek Links	0.6669	0.6558	0.9976	0.9961	0.7988	0.7904
RandomForest	SMOTE + ENN	0.6576	0.703	0.9997	0.9958	0.7922	0.8236
RandomForest	SMOTE + Tomek Links	0.6455	0.666	0.9997	0.999	0.7841	0.799
KNeighbors	SMOTE + ENN	0.5073	0.7965	1.0	0.9505	0.673	**0.8667**
KNeighbors	SMOTE + Tomek Links	0.5112	0.7745	1.0	0.9613	0.676	0.8577

Tabla 33. Métricas CSIC HTTP 2010 con oversampling y undersampling en 1%.

Classifier	Oversampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE + ENN	0.7506	0.7785	0.9934	0.9783	0.8551	0.867
DecisionTree	SMOTE + Tomek Links	0.7698	0.7614	0.9936	0.9927	**0.8674**	0.8618
RandomForest	SMOTE + ENN	0.7243	0.7651	0.9989	0.9857	0.8397	0.8615
RandomForest	SMOTE + Tomek Links	0.733	0.7472	0.9986	0.9969	0.8454	0.8541
KNeighbors	SMOTE + ENN	0.6476	0.8599	0.9997	0.9254	0.786	**0.8914**

KNeighbors	SMOTE Tomek Links	+	0.6523	0.8271	0.9998	0.9515	0.7895	0.8849

Tabla 34. Métricas CSIC HTTP 2010 con oversampling y undersampling en 3.22%.

Classifier	Oversampler		Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE ENN	+	0.7736	0.8164	0.9887	0.9634	0.868	0.8838
DecisionTree	SMOTE Tomek Links	+	0.7936	0.7926	0.989	0.9872	**0.8806**	0.8793
RandomForest	SMOTE ENN	+	0.7579	0.8121	0.9974	0.9758	0.8613	0.8864
RandomForest	SMOTE Tomek Links	+	0.7576	0.7711	0.9973	0.9945	0.8611	0.8686
KNeighbors	SMOTE ENN	+	0.6903	0.8914	0.9955	0.9174	0.8152	**0.9042**
KNeighbors	SMOTE Tomek Links	+	0.7054	0.864	0.9954	0.9411	0.8256	0.9009

Tabla 35. Métricas CSIC HTTP 2010 con oversampling y undersampling en 6.45%.

Classifier	Oversampler		Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE ENN	+	0.8157	0.8693	0.9794	0.9345	0.8901	0.9007
DecisionTree	SMOTE Tomek Links	+	0.8212	0.8215	0.9806	0.9769	**0.8938**	0.8925
RandomForest	SMOTE ENN	+	0.7889	0.8762	0.993	0.9406	0.8792	0.9072
RandomForest	SMOTE Tomek Links	+	0.7869	0.8003	0.9932	0.989	0.8781	0.8847
KNeighbors	SMOTE ENN	+	0.7471	0.9236	0.9956	0.8891	0.8536	**0.908**
KNeighbors	SMOTE Tomek Links	+	0.7386	0.8854	0.9855	0.928	0.8444	0.9062

Tabla 36. Métricas CSIC HTTP 2010 con oversampling y undersampling en 12.91%.

Classifier	Oversampler		Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE ENN	+	0.8605	0.9123	0.963	0.8934	**0.9089**	0.9027
DecisionTree	SMOTE Tomek Links	+	0.8606	0.8558	0.9631	0.961	**0.9089**	0.9053

RandomForest	SMOTE ENN	+	0.8431	0.9329	0.9784	0.8801	0.9057	0.9057
RandomForest	SMOTE Tomek Links	+	0.8422	0.8578	0.9791	0.9705	0.9055	0.9107
KNeighbors	SMOTE ENN	+	0.8106	0.9429	0.9573	0.8653	0.8778	0.9024
KNeighbors	SMOTE Tomek Links	+	0.8106	0.9098	0.9573	0.9119	0.8778	**0.9109**

Tabla 37. Métricas CSIC HTTP 2010 con oversampling y undersampling en 25.82%.

F.2 UNSW-NB15

Classifier	Oversampler		Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE ENN	+	0.9125	0.942	0.9992	0.9971	**0.9539**	0.9687
DecisionTree	SMOTE Tomek Links	+	0.9117	0.9203	0.9992	0.9985	0.9534	0.9578
RandomForest	SMOTE ENN	+	0.9068	0.9546	0.9998	0.9969	0.951	**0.9753**
RandomForest	SMOTE Tomek Links	+	0.9056	0.9323	0.9998	0.9986	0.9503	0.9643
KNeighbors	SMOTE ENN	+	0.7945	0.9317	0.9997	0.9903	0.8854	0.9601
KNeighbors	SMOTE Tomek Links	+	0.7947	0.9262	0.9997	0.9915	0.8855	0.9577

Tabla 38. Métricas UNSW-NB15 con oversampling y undersampling en 1%.

Classifier	Oversampler		Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE ENN	+	0.9199	0.956	0.9988	0.9959	0.9577	0.9755
DecisionTree	SMOTE Tomek Links	+	0.9219	0.9334	0.9989	0.998	**0.9588**	0.9646
RandomForest	SMOTE ENN	+	0.9185	0.968	0.9996	0.9957	0.9573	**0.9817**
RandomForest	SMOTE Tomek Links	+	0.9154	0.9442	0.9996	0.9979	0.9556	0.9703
KNeighbors	SMOTE ENN	+	0.8174	0.9463	0.9995	0.9893	0.8993	0.9673
KNeighbors	SMOTE Tomek Links	+	0.8176	0.9395	0.9995	0.9906	0.8994	0.9643

Tabla 39. Métricas UNSW-NB15 con oversampling y undersampling en 1.58%.

Classifier	Oversampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE + ENN	0.9355	0.9725	0.9982	0.9939	0.9658	0.9831
DecisionTree	SMOTE + Tomek Links	0.9347	0.948	0.9982	0.997	0.9654	0.9719
RandomForest	SMOTE + ENN	0.9347	0.9812	0.9992	0.9935	**0.9659**	**0.9873**
RandomForest	SMOTE + Tomek Links	0.9348	0.9611	0.9992	0.9968	**0.9659**	0.9787
KNeighbors	SMOTE + ENN	0.8556	0.9611	0.9987	0.9876	0.9216	0.9742
KNeighbors	SMOTE + Tomek Links	0.855	0.9544	0.9986	0.9891	0.9213	0.9714

Tabla 40. Métricas UNSW-NB15 con oversampling y undersampling en 3.16%.

Classifier	Oversampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE + ENN	0.9512	0.9828	0.9972	0.9922	0.9736	0.9875
DecisionTree	SMOTE + Tomek Links	0.9511	0.9626	0.9972	0.9956	0.9736	0.9788
RandomForest	SMOTE + ENN	0.9537	0.9904	0.9982	0.9911	0.9754	**0.9907**
RandomForest	SMOTE + Tomek Links	0.9539	0.9742	0.9982	0.9956	**0.9756**	0.9847
KNeighbors	SMOTE + ENN	0.8944	0.9747	0.9968	0.9856	0.9428	0.9801
KNeighbors	SMOTE + Tomek Links	0.8946	0.9687	0.9968	0.9875	0.9429	0.978

Tabla 41. Métricas UNSW-NB15 con oversampling y undersampling en 6.32%.

Classifier	Oversampler	Recall Normal	Recall	TNR Normal	TNR	MRTN Normal	MRTN
DecisionTree	SMOTE + ENN	0.9678	0.9923	0.9954	0.9899	0.9815	0.9911
DecisionTree	SMOTE + Tomek Links	0.9679	0.9767	0.9955	0.9941	0.9815	0.9853
RandomForest	SMOTE + ENN	0.9736	0.996	0.9962	0.9887	**0.9847**	**0.9923**
RandomForest	SMOTE + Tomek Links	0.9736	0.9853	0.9962	0.9937	**0.9847**	0.9895
KNeighbors	SMOTE + ENN	0.9334	0.982	0.9935	0.9841	0.9625	0.9831

KNeighbors	SMOTE + Tomek Links	0.9334	0.9788	0.9935	0.9858	0.9625	0.9823

Tabla 42. Métricas UNSW-NB15 con oversampling y undersampling en 12.64%.

G Importancia de Gini para los atributos del Árbol de Decisión en la base de datos UNSW-NB15

A continuación, se muestran los atributos del conjunto de datos ordenados por su importancia de Gini normalizada obtenidos de un árbol de decisión entrenado. Aparecen remarcados en negrita los atributos seleccionados (con importancia mayor al 0.4%).

- **(0.8618802404526646, 'sttl')**
- **(0.03032850498987743, 'Sload')**
- **(0.022356993678468544, 'ct_srv_dst')**
- **(0.019480995521036104, 'sbytes')**
- **(0.012042467575304458, 'smeansz')**
- **(0.004885052556334711, 'Spkts')**
- **(0.004046390543690712, 'synack')**
- (0.0039958307802971885, 'dbytes')
- (0.0038782029332558427, 'ct_dst_src_ltm')
- (0.003579986992987102, 'ct_srv_src')
- (0.002955596720254618, 'Sintpkt')
- (0.0028431526726990955, 'dmeansz')
- (0.0025883979611259463, 'tcprtt')
- (0.0025459265882051343, 'dtcpb')
- (0.0024661087262510964, 'stcpb')
- (0.0021521558966897275, 'ackdat')
- (0.0018899159881252334, 'Djit')
- (0.00186137942594, 'dur')
- (0.0016785673915884819, 'Sjit')
- (0.0015901456480895083, 'Dload')
- (0.0015362631940688217, 'ct_src_ ltm')
- (0.001414180411790317, 'Dintpkt')
- (0.001343224760503308, 'sloss')
- (0.0012932642109205722, 'ct_dst_ltm')
- (0.0009070051333182579, 'swin')
- (0.0008329849748089569, 'ct_src_dport_ltm')
- (0.0007338859986131738, 'trans_depth')
- (0.0006201715160396678, 'ct_dst_sport_ltm')
- (0.0005972977112583879, 'Dpkts')
- (0.0003517563805906093, 'ct_flw_http_mthd')
- (0.0003281657068169071, 'res_bdy_len')
- (0.0003270585668421103 3, 'ct_ftp_cmd')
- (0.0002592876012821659 6, 'dloss')
- (0.0002460066574714213 3, 'is_ftp_login')
- (0.0001400148753213279 7, 'dttl')
- (0.0, 'dwin')

H Resultados de la métrica MRTN con distintos umbrales

Las figuras a continuación muestran las curvas ROC para los algoritmos de árboles de decisión y vecinos próximos utilizando la base de datos CSIC HTTP 2010. En ellas se puede observar como para el algoritmo de árboles de decisión el punto óptimo según MRTN no está muy lejos del punto elegido por el algoritmo, pero sin embargo, en el algoritmo de vecinos próximos sí que se aprecia una distancia mucho mayor.

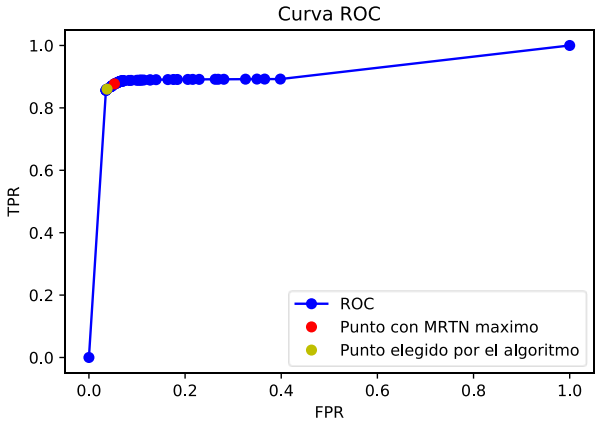

Figura 11. Curva ROC Árboles de decisión.

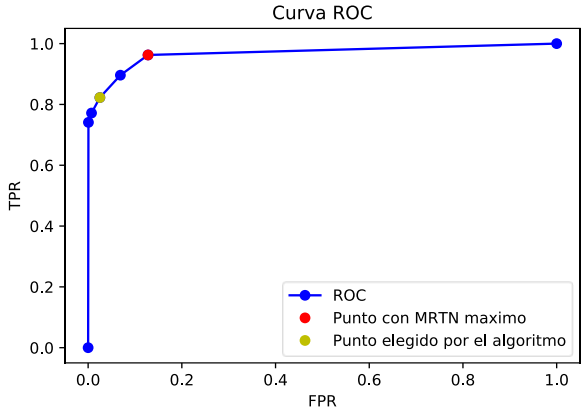

Figura 12. Curva ROC Vecinos Próximos.

I Hiperparámetros del experimento con MRTN

Los hiperparámetros de cada uno de los ejemplos mostrados en la Tabla 2 son los siguientes:

- Ejemplo 1: LocalOutlierFactor(leaf_size=10, metric='braycurtis', n_neighbors=3, algorithm=’auto’, contamination=0.2582, n_jobs=-1, p=2, metric_params=None, novelty=True)
- Ejemplo 2: LocalOutlierFactor(leaf_size=10, metric='correlation', n_neighbors=15, algorithm=’auto’, contamination=0.2582, n_jobs=-1, p=2, metric_params=None, novelty=True)
- Ejemplo 3: LocalOutlierFactor(leaf_size=10, metric='chebyshev', n_neighbors=3, algorithm=’auto’, contamination=0.2582, n_jobs=-1, p=2, metric_params=None, novelty=True)

- Ejemplo 4: LocalOutlierFactor(leaf_size=10, metric='rogerstanimoto', n_neighbors=15 , algorithm=’auto’, contamination=0.2582, n_jobs=-1, p=2, metric_params=None, novelty=True)